

Transformations and Fitting

EECS 442 – Jeong Joon Park
Winter 2024, University of Michigan

Administrivia (Project)

- We expect:
- That you explore a problem that's challenging
~ one HW worth of work per person
- You to write a large fraction of the code
yourself with the help of libraries like Pytorch,
Numpy, etc.
- That the projects are a bit more open ended
than the homeworks
- That you get some experience with doing a
project start to finish, from developing the
method to evaluating it and writing it up.

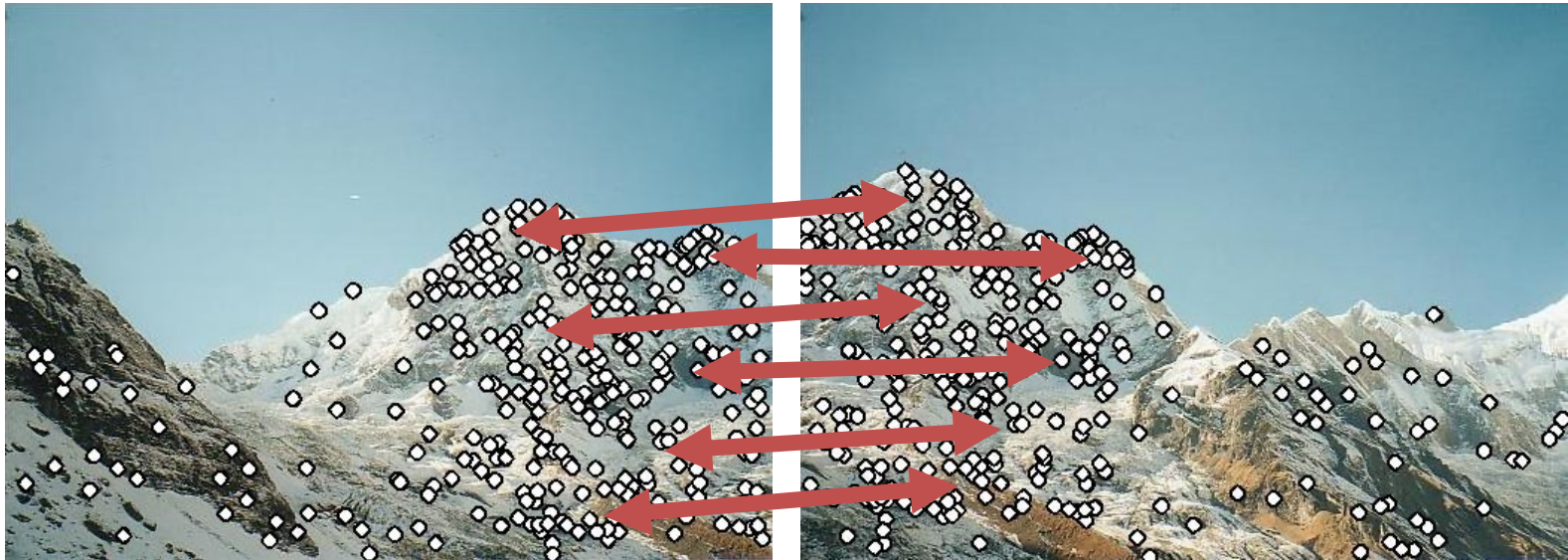
Administrivia (Project)

- We do not expect:
- A course project that is worthy of a publication
- A new idea (implementing an old idea is totally fine!)
- That your project works (so long as you can identify and diagnose why the system doesn't work)

Administrivia (Project)

- Example project ideas on the website.
- Either choose the example idea or your own!
- Timelines listed on the website schedule
- 3~4 group members: Use Piazza to find!

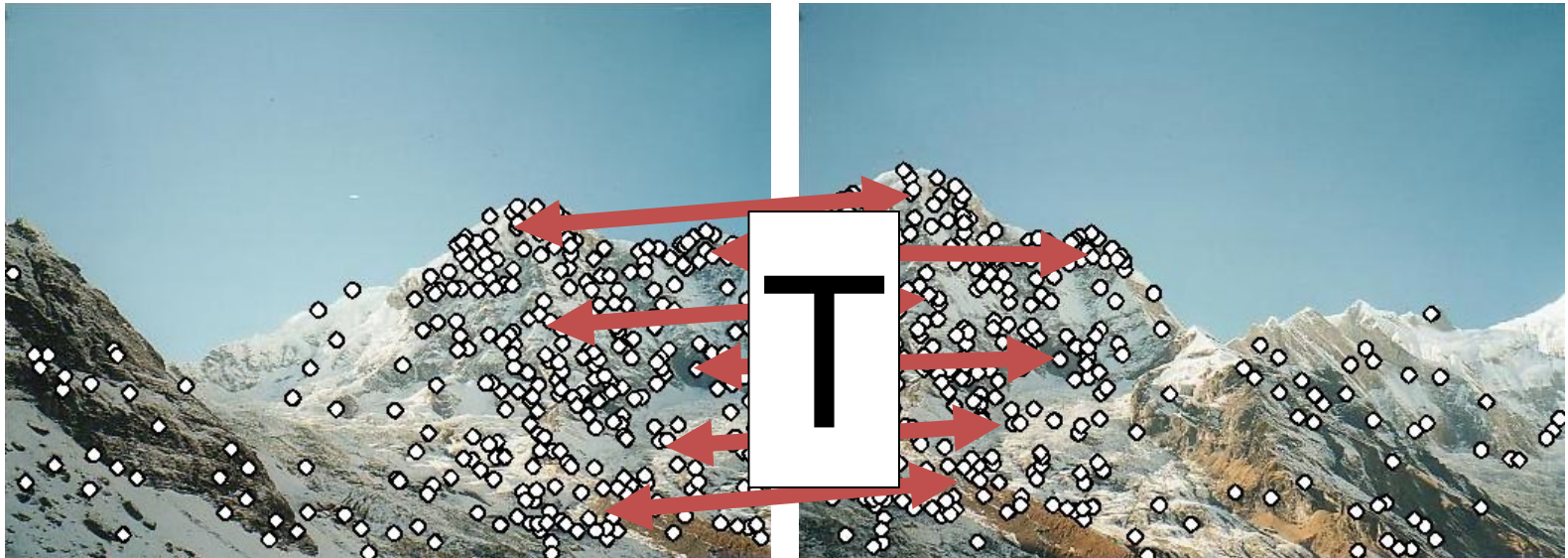
Last Class



1. How do we find distinctive / easy to locate features? (*Harris/Laplacian of Gaussian*)
2. How do we describe the regions around them? (*Normalize window, use histogram of gradient orientations*)

Earlier I promised

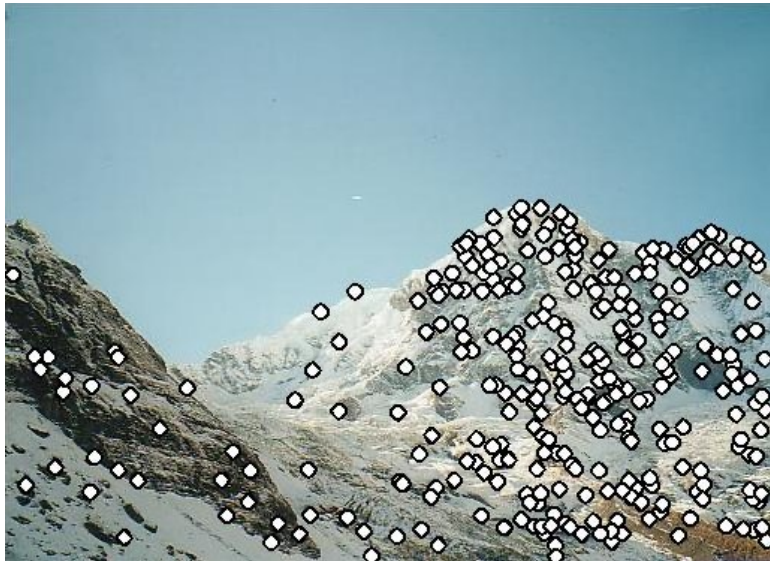
Solving for a Transformation



3: Solve for transformation T (e.g. such that $\mathbf{p1} \equiv T \mathbf{p2}$) that fits the matches well

Before Anything Else, Remember

You, with your
gigantic brain, see:



The computer
sees:

097	097	097	097	097	097	097	097	097	096	097	097	096	096	096
100	100	100	100	100	100	101	101	102	101	100	100	100	099	
105	105	105	105	105	105	105	103	102	102	101	103	104	105	
109	109	109	109	109	110	107	118	145	132	120	112	106	103	
113	113	113	112	112	113	110	129	160	160	164	162	157	151	
118	117	118	123	119	118	112	125	142	134	135	139	139	175	
123	121	125	162	166	157	149	153	160	151	150	146	137	168	
127	127	125	168	147	117	139	135	126	147	147	149	156	160	
133	130	150	179	145	132	160	134	150	150	111	145	126	121	
138	134	179	185	141	090	166	117	120	153	111	153	114	126	
144	151	188	178	159	154	172	147	159	170	147	185	105	122	
152	157	184	183	142	127	141	133	137	141	131	147	144	147	
130	147	185	180	139	131	154	121	140	147	107	147	120	128	
035	102	194	175	149	140	179	128	146	168	096	163	101	125	

You should expect **noise** (not at quite the right pixel) and **outliers** (random matches)

Today

- How do we fit **models** (i.e., a parametric representation of data that's smaller than the data) to data?
- How do we handle:
 - **Noise** – least squares / total least squares
 - **Outliers** – RANSAC (random sample consensus)
 - **Multiple models** – Hough Transform (can also make RANSAC handle this with some effort)

Working Example: Lines

- We'll handle lines as our **models** today since you are more familiar with them than others
- Next class will cover more complex models. I promise we'll eventually stitch images together
- You can apply today's techniques on next class's models

Model Fitting

Need three ingredients

Data: what data are we trying to explain with a model?

Model: what's the compressed, parametric form of the data?

Objective function: given a prediction, how do we evaluate how correct it is?

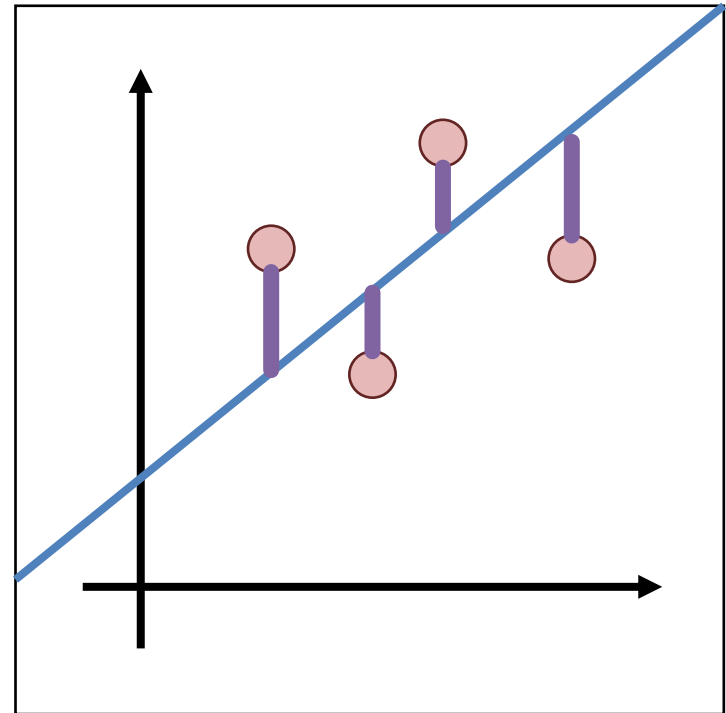
Example: Least-Squares

Fitting a line to data

Data: $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$

Model: (m, b) $y_i = mx_i + b$
Or (\mathbf{w}) $y_i = \mathbf{w}^T \mathbf{x}_i$

Objective function:
 $(y_i - \mathbf{w}^T \mathbf{x}_i)^2$



Least-Squares Setup

$$\sum_{i=1}^k (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \quad \rightarrow \quad \|\mathbf{Y} - \mathbf{X}\mathbf{w}\|_2^2$$

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_1 & 1 \\ \vdots & 1 \\ x_k & 1 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} m \\ b \end{bmatrix}$$

Solving Least-Squares

$$\|Y - Xw\|_2^2$$

Where can I find derivatives + matrix expressions and matrix identities?



The Matrix Cookbook



Kaare Brandt Petersen
Michael Syskind Pedersen

VERSION: NOVEMBER 15, 2012

Then our objective function is find the minimum of the loss function.

Solving Least-Squares

$$\begin{aligned} & \|Y - Xw\|_2^2 \quad \text{we assume it's:} \\ & \quad \text{Convex function!} \\ & = Y^2 - 2Y^T Xw + X^T X w^2 \quad \text{So local minimum} \rightarrow \text{global} \\ & \quad \text{minimum.} \\ \frac{\partial}{\partial w} \|Y - Xw\|_2^2 &= 2X^T Xw - 2X^T Y \quad \frac{\partial}{\partial w} \Rightarrow -2X^T Y + 2X^T Xw \end{aligned}$$

Recall: derivative is 0 at a maximum / minimum. Same is true about gradients.

$$0 = 2X^T Xw - 2X^T Y$$

$$X^T Xw = X^T Y$$

$$w = (X^T X)^{-1} X^T Y$$

Aside: **0** is a vector of 0s. **1** is a vector of 1s.

Two Solutions to Getting W

In One Go

Implicit form
(normal equations)

$$X^T X w = X^T Y$$

Explicit form
(don't do this)

$$w = (X^T X)^{-1} X^T Y$$

Iteratively

Recall: gradient is also direction that makes function go up the most.

What could we do?

$$w_0 = 0$$

← initial w₀ value first

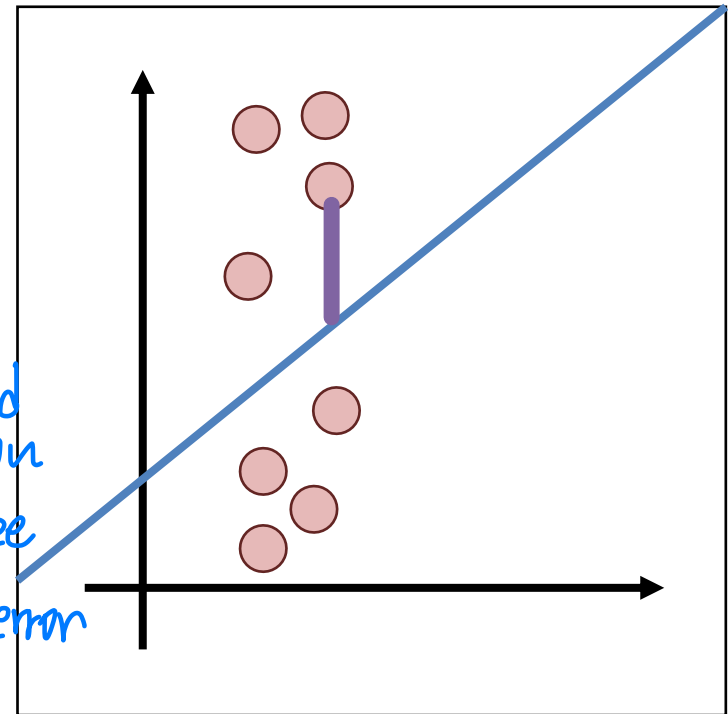
$$w_{i+1} = w_i - \gamma \left(\frac{\partial}{\partial w} \|Y - Xw\|_2^2 \right)$$

iteration and update the w.

What's The Problem?

使用 slope 和 y 轴交点的坏处.

- Vertical lines impossible!
 - Not rotationally invariant: the line will change depending on orientation of points
 - Hint: y-axis distance
- if rotate the points around and fit it again, then you might see a different error



Alternate Formulation

Important part: Any line can be framed in terms of normal \mathbf{n} and offset d

$$\mathbf{n} = [a, b]$$

Now: $ax + by - d = 0$

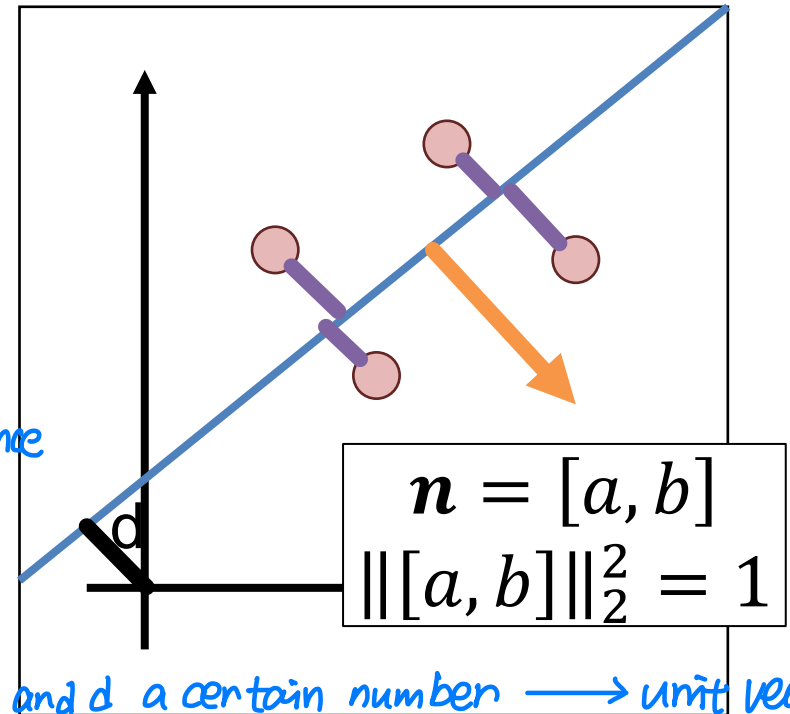
$$\mathbf{n}^T [x, y] - d = 0$$

$$\mathbf{n}^T = \begin{bmatrix} a \\ b \end{bmatrix} \quad \begin{bmatrix} a \\ b \end{bmatrix} [x, y] = ax + by$$

Point to line distance:

$$\frac{\mathbf{n}^T [x, y] - d}{\|\mathbf{n}\|_2^2} = \boxed{\mathbf{n}^T [x, y] - d}$$

How to calculate the distance from a point to a line.



Also normalize the norm vector. By dividing a, b and d a certain number. \rightarrow unit vector.

Total Least-Squares

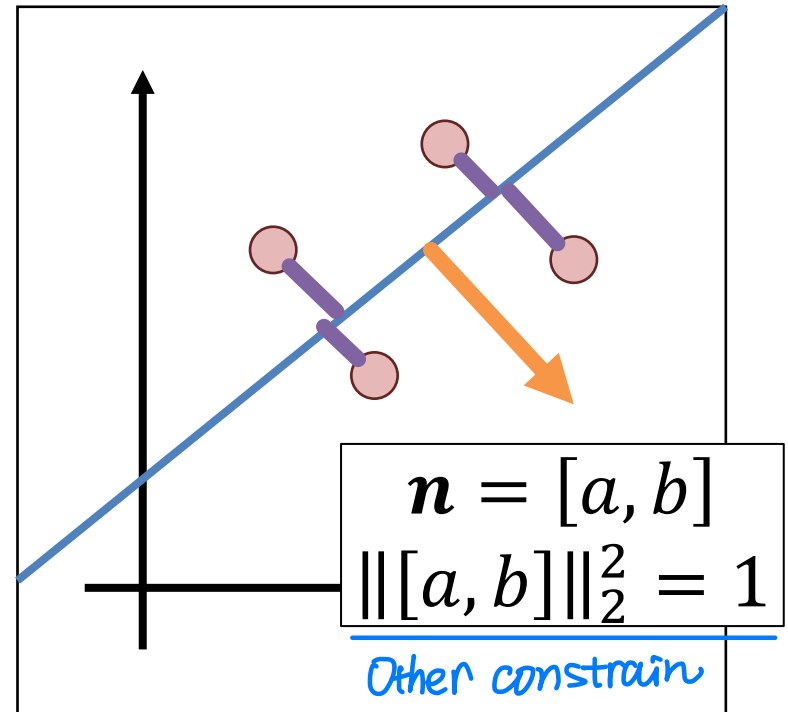
Fitting a line to data

Data: $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$

Model: $(\mathbf{n}, d), \|\mathbf{n}\|^2 = 1$
 $\mathbf{n}^T[x_i, y_i] - d = 0$

what we want

Objective function:
 $(\mathbf{n}^T[x_i, y_i] - d)^2$



Total Least Squares Setup

Figure out objective first, then figure out $\|n\|=1$

$$\sum_{i=1}^k (\mathbf{n}^T [\mathbf{x}, \mathbf{y}] - d)^2 \rightarrow \|\mathbf{X}\mathbf{n} - \mathbf{1}d\|_2^2$$

$$\mathbf{X} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_k & y_k \end{bmatrix} \quad \mathbf{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \quad \mathbf{n} = \begin{bmatrix} a \\ b \end{bmatrix} \quad \boldsymbol{\mu} = \frac{1}{k} \mathbf{1}^T \mathbf{X} \begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix}$$


The mean / center of mass of the points:
`np.sum(X,axis=0)`. We'll use it later

Total Least Squares Setup

Want to make sure that the following is minimized:

$$\|\mathbf{X}\mathbf{n} - \mathbf{1}d\|_2^2$$

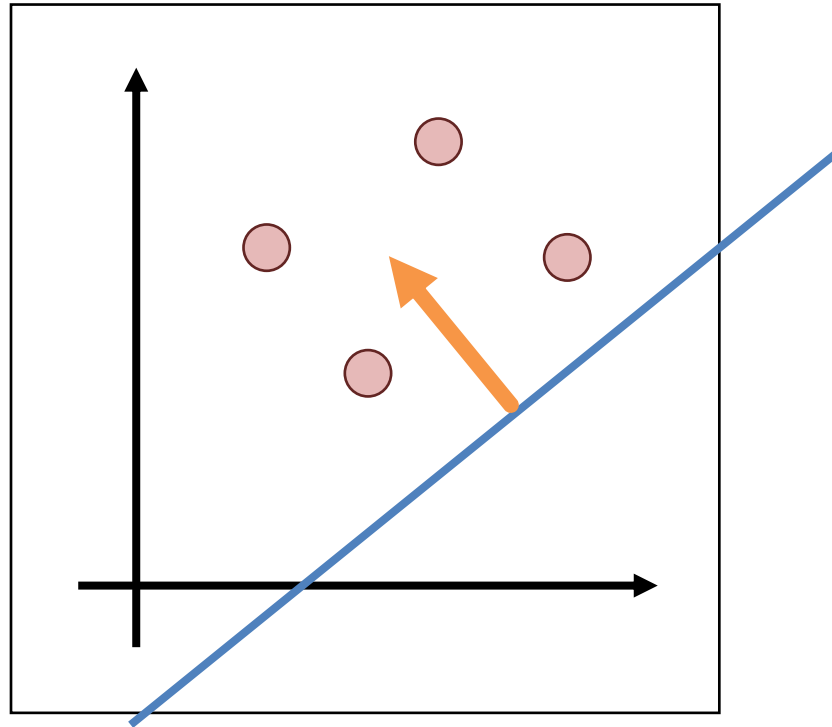
Won't derive, but can show that whenever you find the \mathbf{n} , and d that minimize the objective, $d = \boldsymbol{\mu}\mathbf{n}$.
(at back of slides if you're curious.)

$$\mathbf{X} = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_k & y_k \end{bmatrix} \quad \mathbf{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \quad \mathbf{n} = \begin{bmatrix} a \\ b \end{bmatrix} \quad \boldsymbol{\mu} = \frac{1}{k} \mathbf{1}^T \mathbf{X}$$


The mean / center of mass of the points:
`np.sum(X,axis=0)`. We'll use it later

Intuition

Once you find the angle (\mathbf{n}), you will need to shift to the center



Solving Total Least-Squares

$$\begin{aligned}\|\mathbf{X}\mathbf{n} - \mathbf{1}d\|_2^2 &= \|\mathbf{X}\mathbf{n} - \mathbf{1}\mu\mathbf{n}\|_2^2 & d &= \mu\mathbf{n} \\ &= \|(\mathbf{X} - \mathbf{1}\mu)\mathbf{n}\|_2^2\end{aligned}$$

Objective is then:

$$\arg \min_{\|\mathbf{n}\|=1} \|(\mathbf{X} - \mathbf{1}\mu)\mathbf{n}\|_2^2$$



\mathbf{n} that makes the expression smallest given constraint

Homogeneous Least Squares

the v that satisfies that equation is

$$\arg \min_{\|v\|_2=1} \|Av\|_2^2 \rightarrow \text{Eigenvector corresponding to smallest eigenvalue of } A^T A$$

Why do we need $\|v\|^2 = 1$ or some other constraint?

Applying it in our case:

$$n = \text{smallest_eigenvec}((X - \mathbf{1}\mu)^T (X - \mathbf{1}\mu))$$
$$n^T [x, y] - \mu n = 0$$

Note: technically homogeneous only refers to $\|Av\|=0$ but it's common shorthand in computer vision to refer to the specific problem of $\|v\|=1$

Details For ML-People

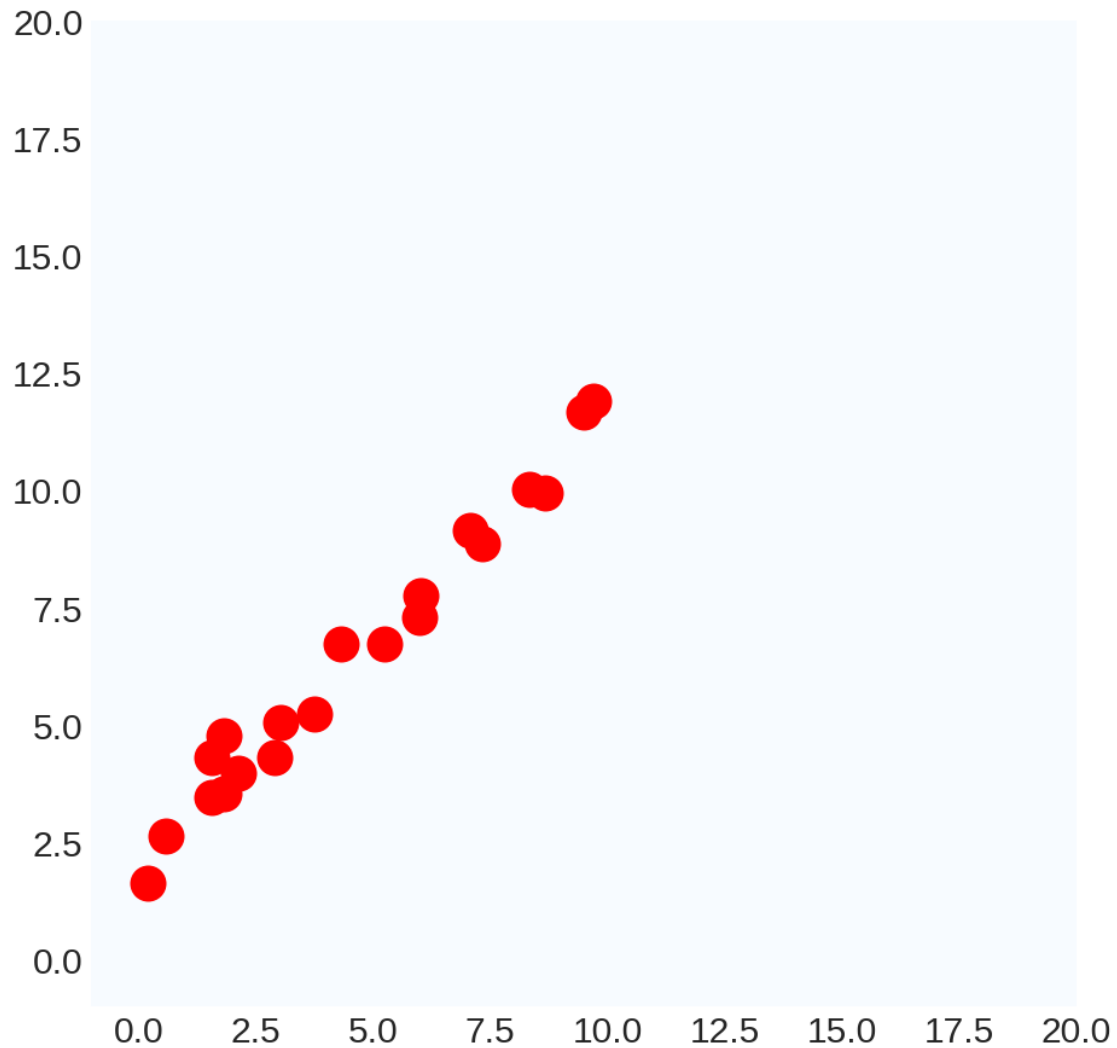
Matrix we take the eigenvector of looks like:

$$(X - \mathbf{1}\mu)^T(X - \mathbf{1}\mu) = \begin{bmatrix} \sum_i (x_i - \mu_x)^2 & \sum_i (x_i - \mu_x)(y_i - \mu_y) \\ \sum_i (x_i - \mu_x)(y_i - \mu_y) & \sum_i (y_i - \mu_y)^2 \end{bmatrix}$$

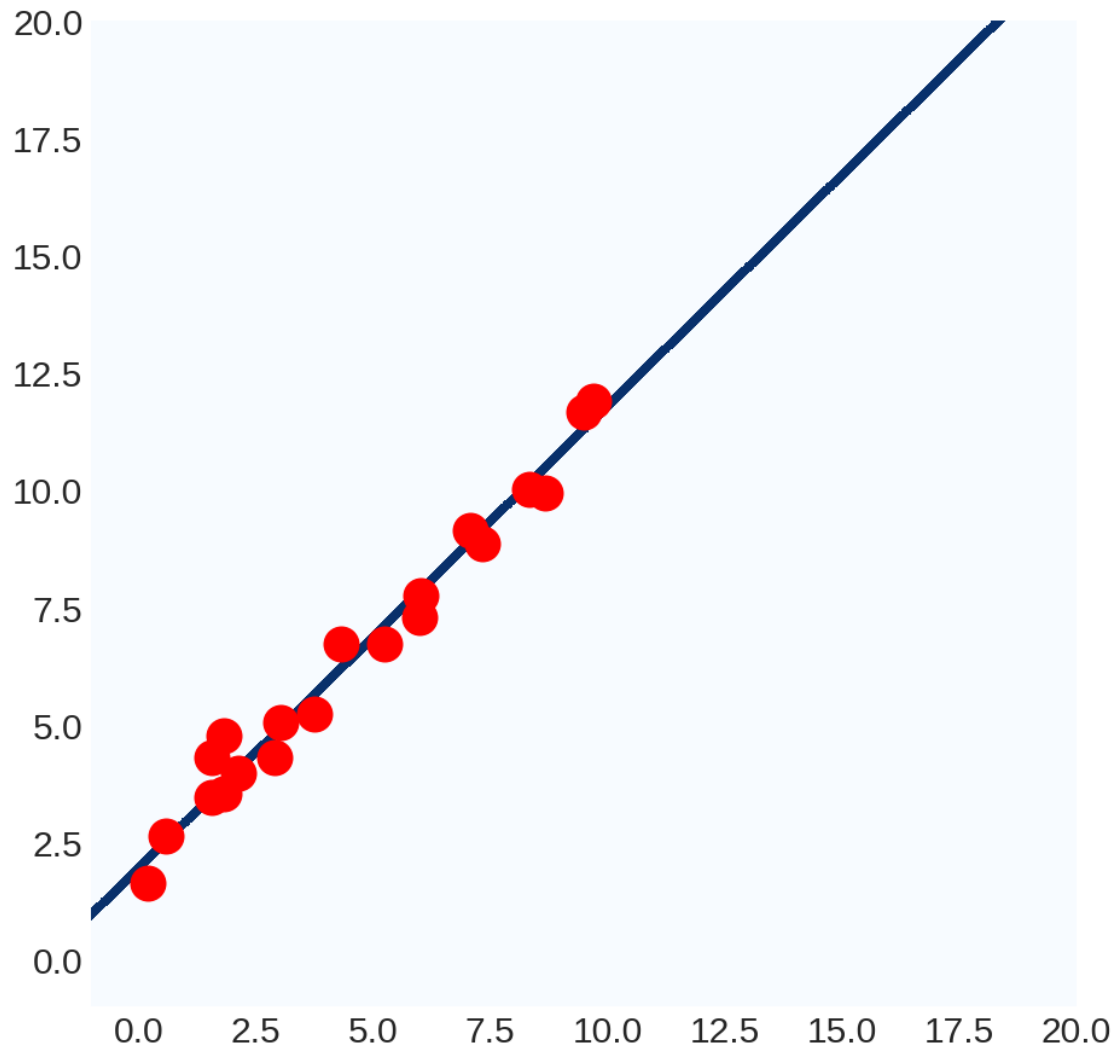
This is a scatter matrix or scalar multiple of the covariance matrix. We're doing PCA, but taking the least principal component to get the normal.

Note: If you don't know PCA, just ignore this slide; it's to help build connections to people with a background in data science/ML.

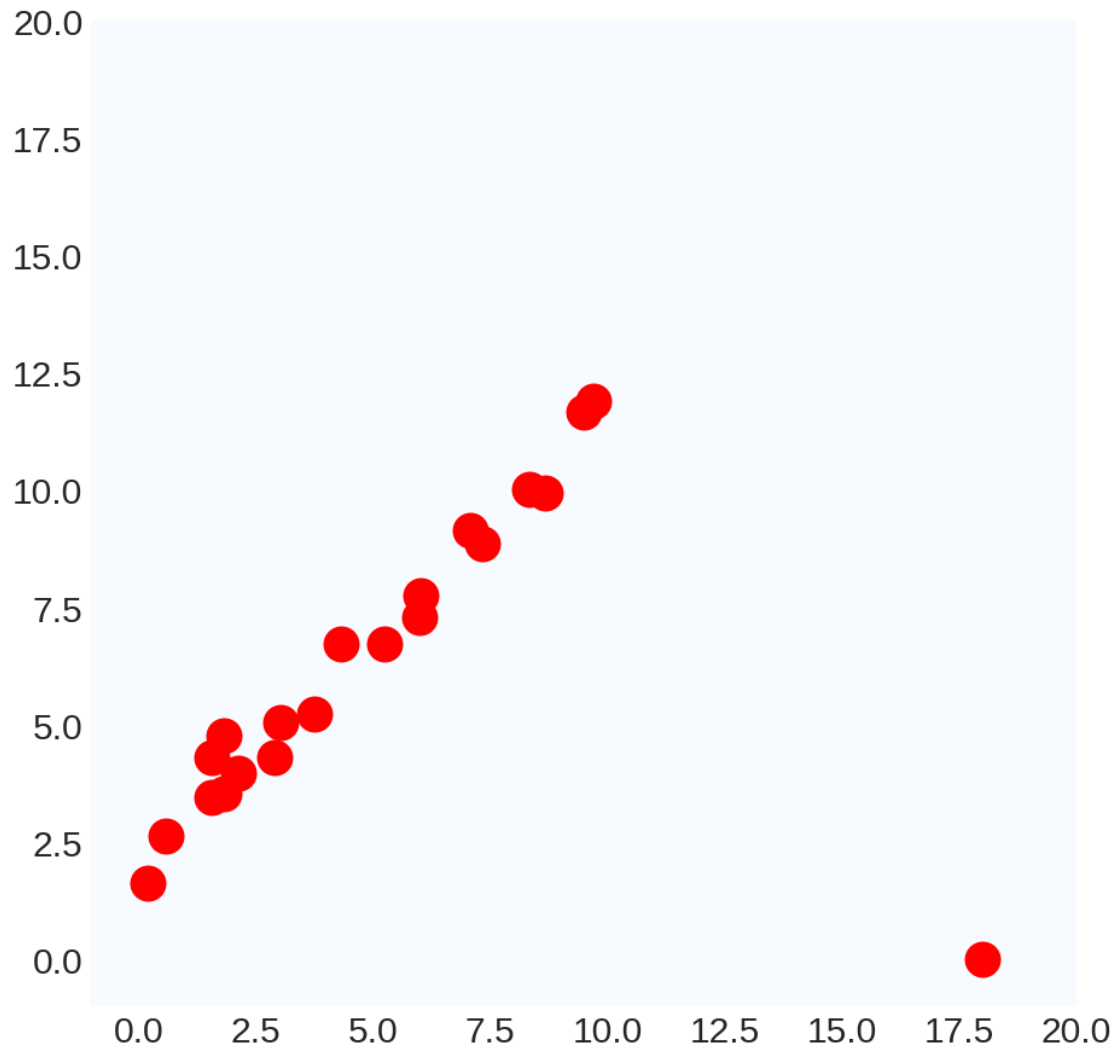
Running Least-Squares



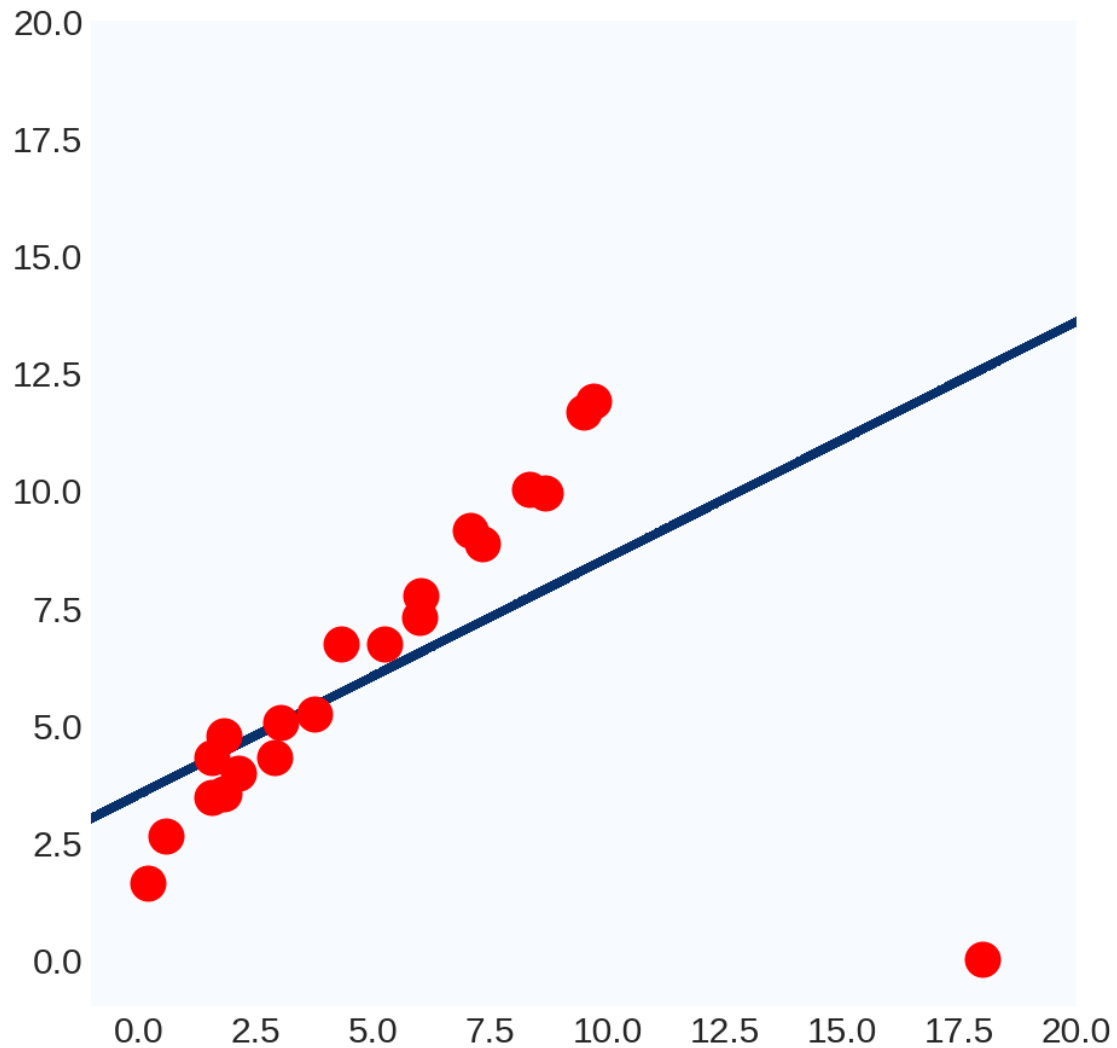
Running Least-Squares



Ruining Least Squares



Ruining Least Squares



Ruining Least Squares

Way to think of it #1:

$$\|Y - Xw\|_2^2$$

$100^2 \gg 10^2$: least-squares prefers having no large errors, even if the model is useless overall

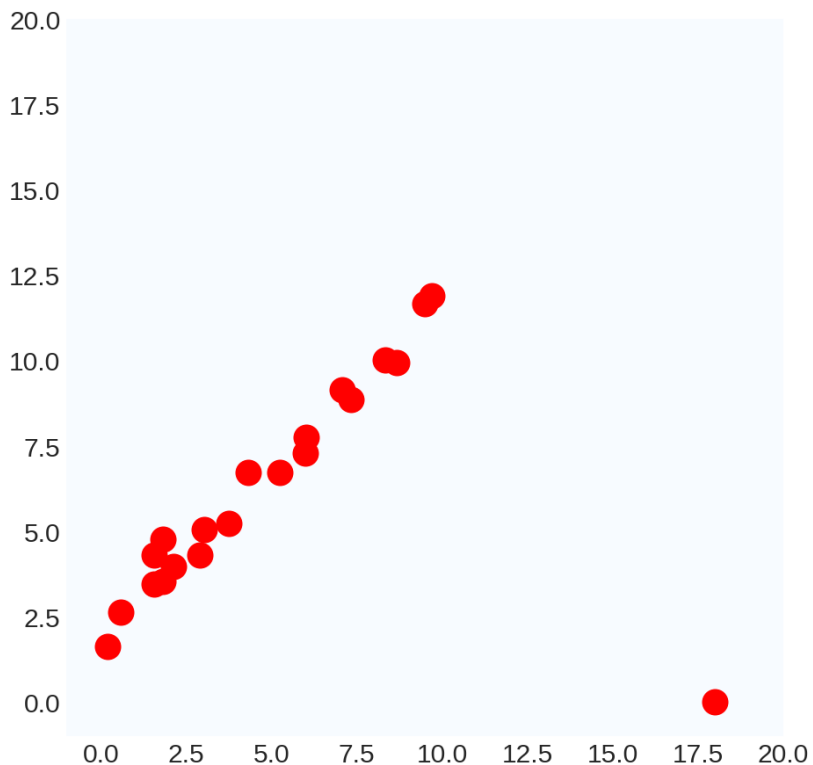
Way to think of it #2:

$$w = \underline{(X^T X)^{-1} X^T Y}$$

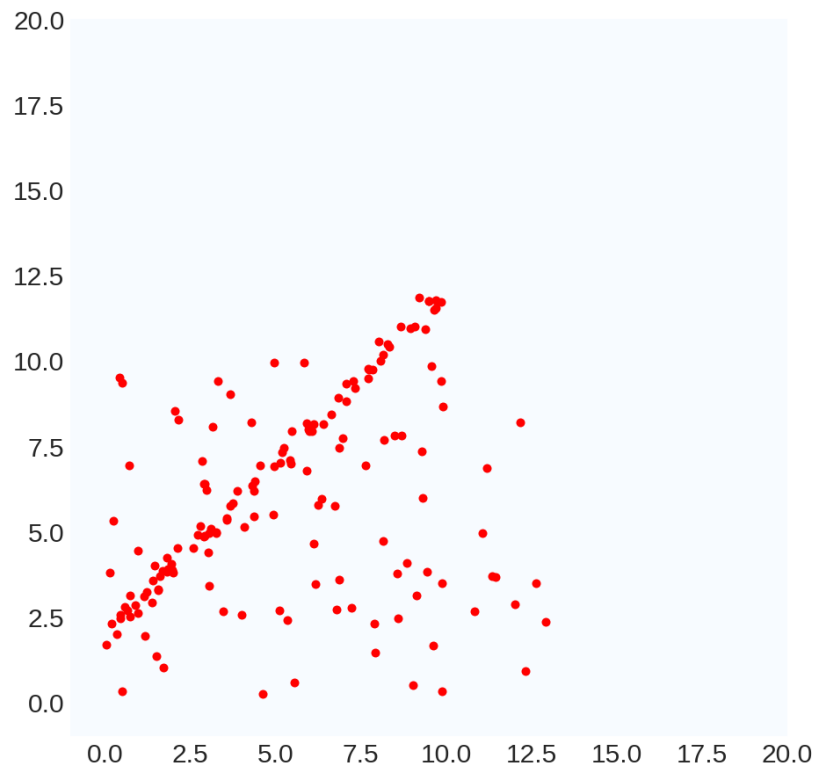
Parameters w are a linear transformation of the output variable: can manipulate w by manipulating X, Y .

Outliers in Computer Vision

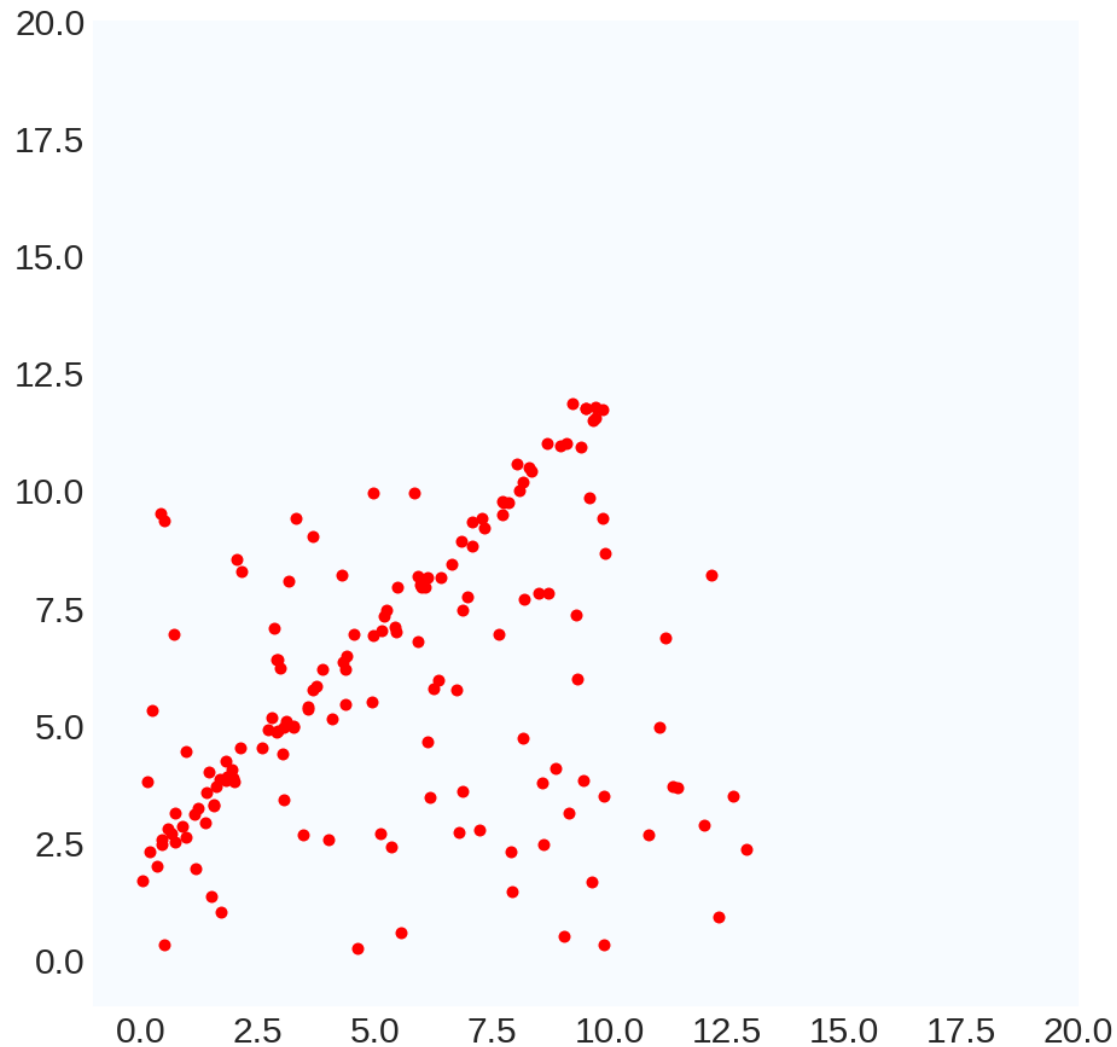
Single outlier:
rare



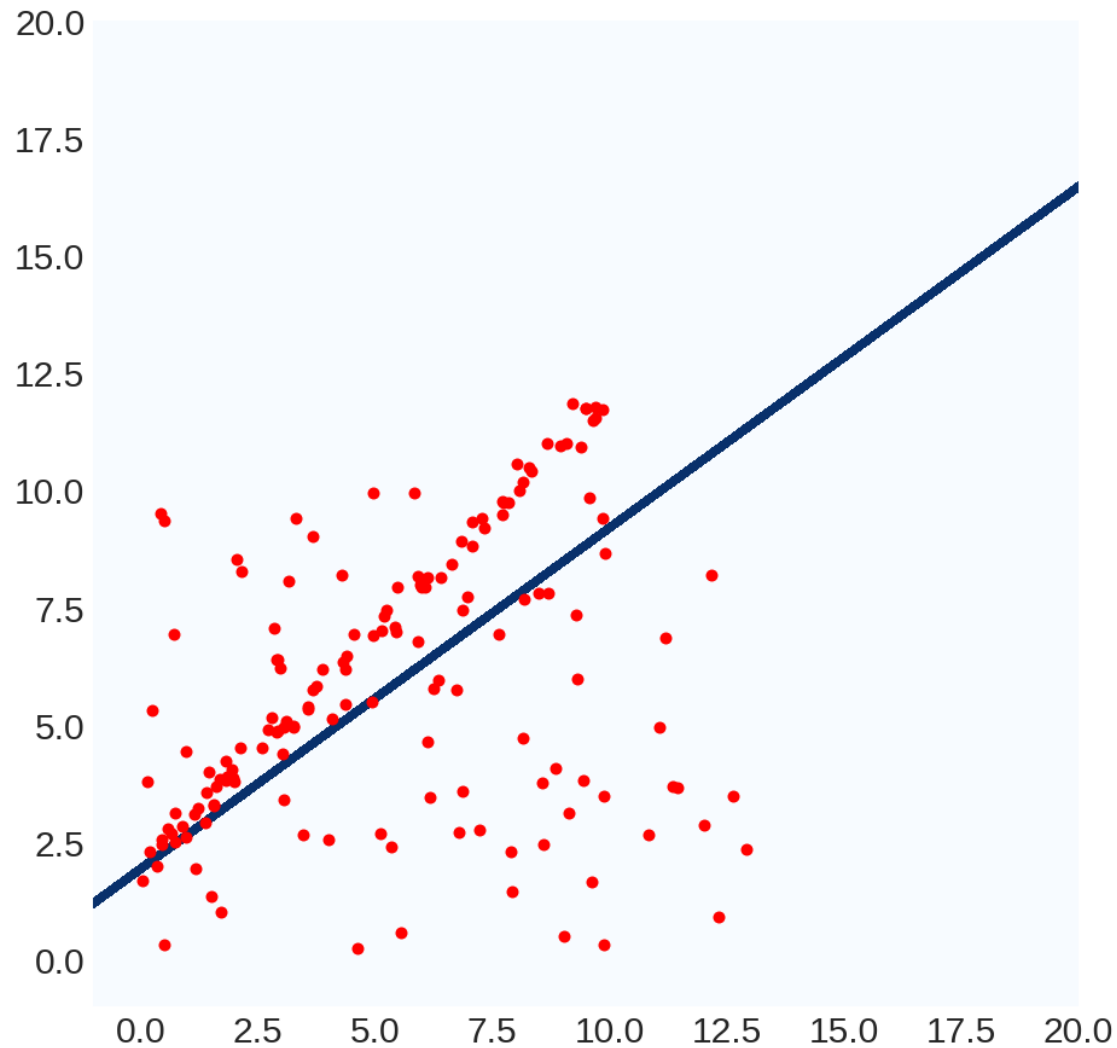
Many outliers:
common



Ruining Least Squares Continued



Ruining Least Squares Continued



A Simple, Yet Clever Idea

- *What we really want*: model explains **many** points “**well**”
- *Least Squares*: model makes as few big mistakes as possible over the entire dataset
- *New objective*: find model for which error is “small” for as many data points as possible
- *Method*: RANSAC (**R**andom **S**ample **C**onsensus)

M. A. Fischler, R. C. Bolles. [Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography](#). Comm. of the ACM, Vol 24, pp 381-395, 1981.

RANSAC For Lines

bestLine, bestCount = None, -1

for trial in range(numTrials):

 subset = pickPairOfPoints(data)

 line = totalLeastSquares(subset)

 E = linePointDistance(data, line)

error(all points)

 inliers = E < threshold

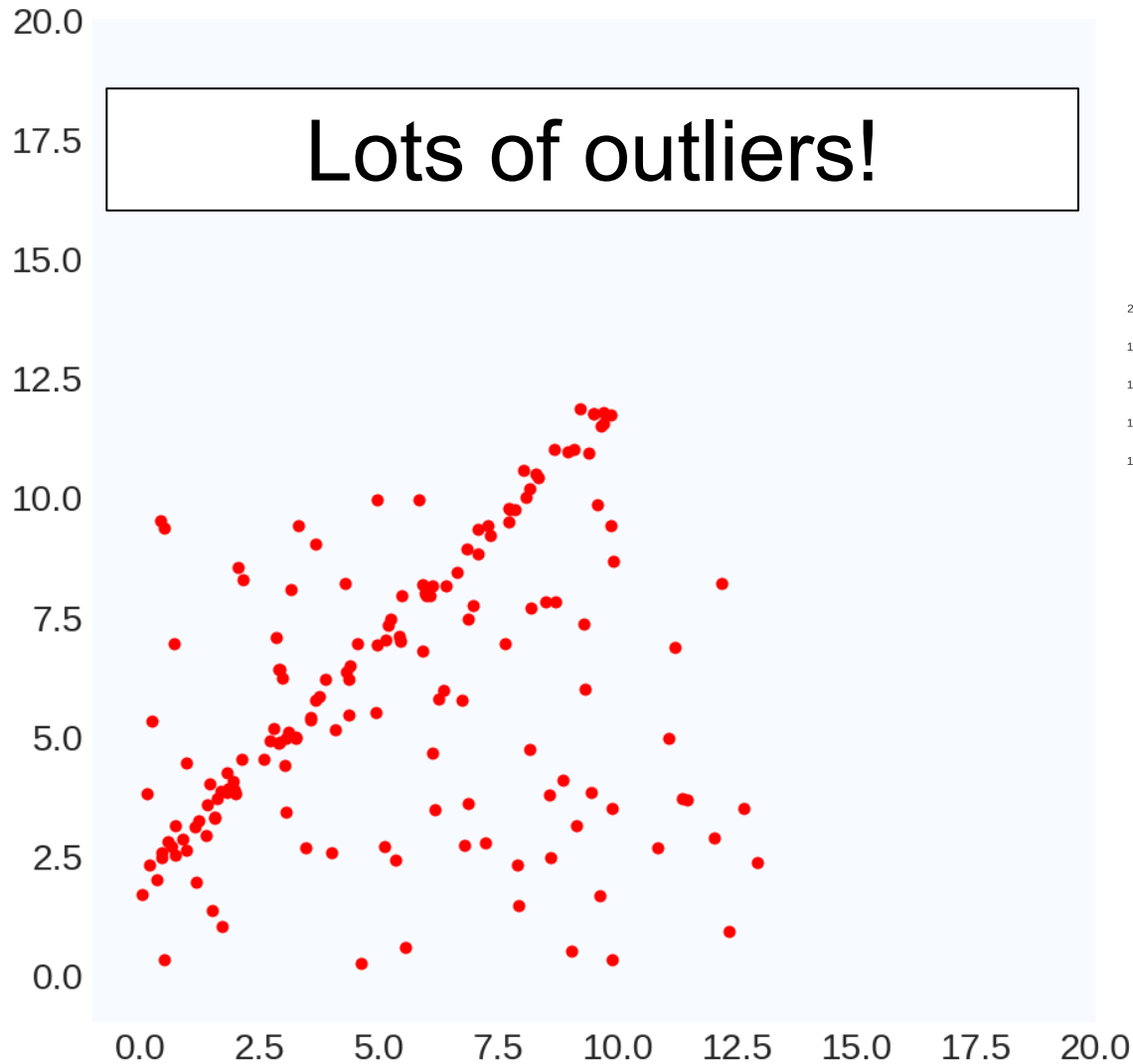
= a vector, hold the error for each point.

 if #inliers > bestCount:

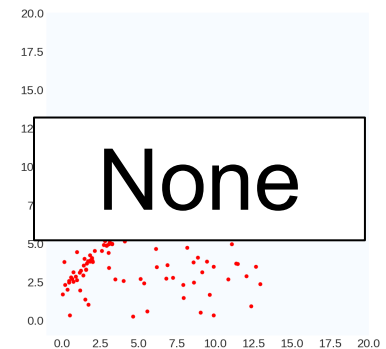
 bestLine, bestCount = line, #inliers

Running RANSAC

Trial
#1



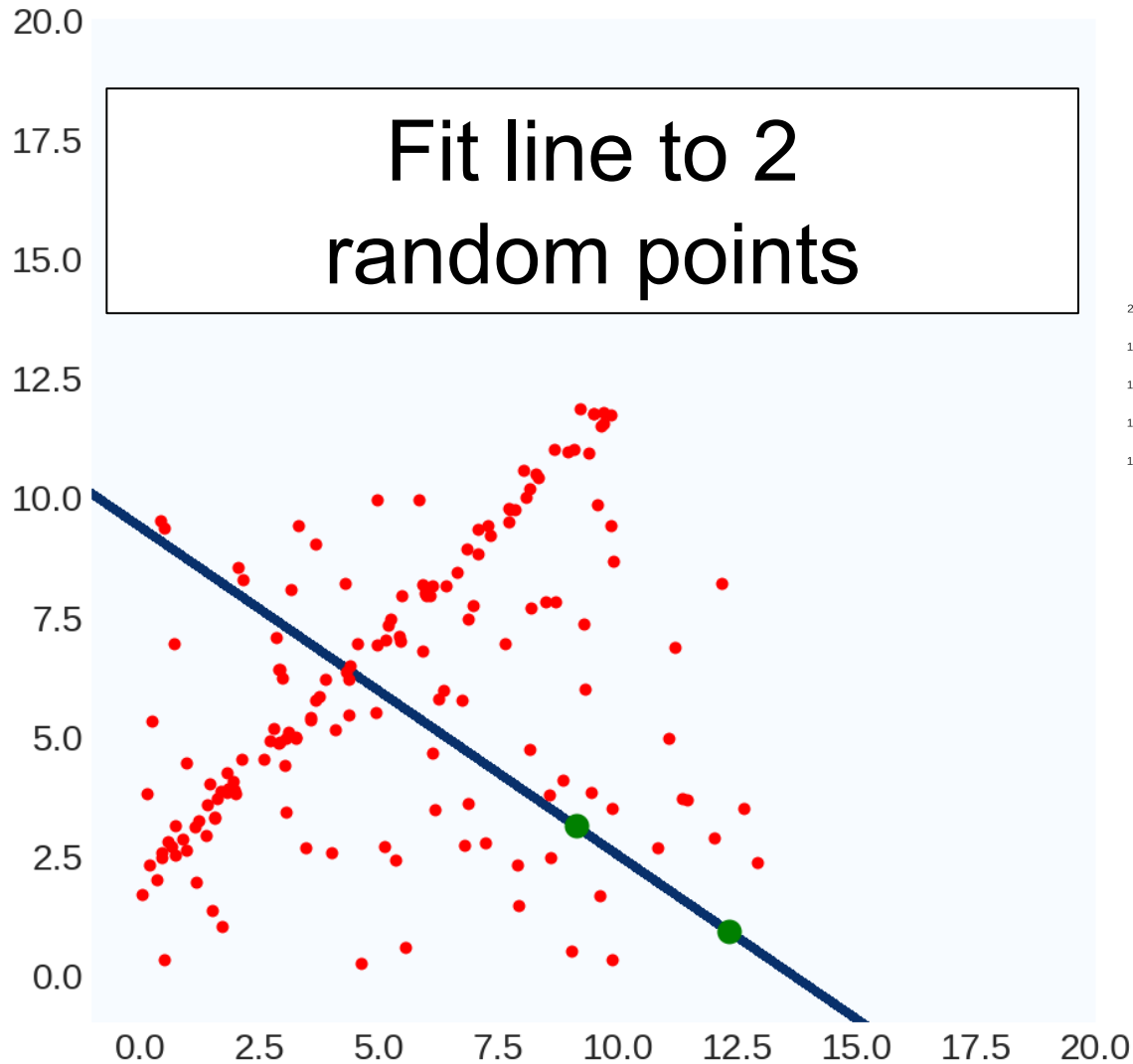
Best
Model:



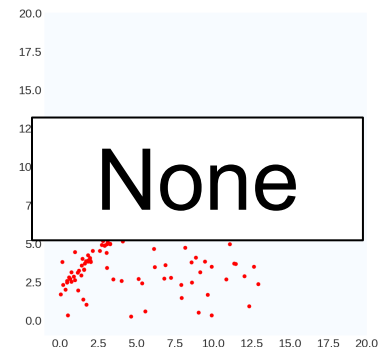
Best
Count:
-1

Running RANSAC

Trial
#1



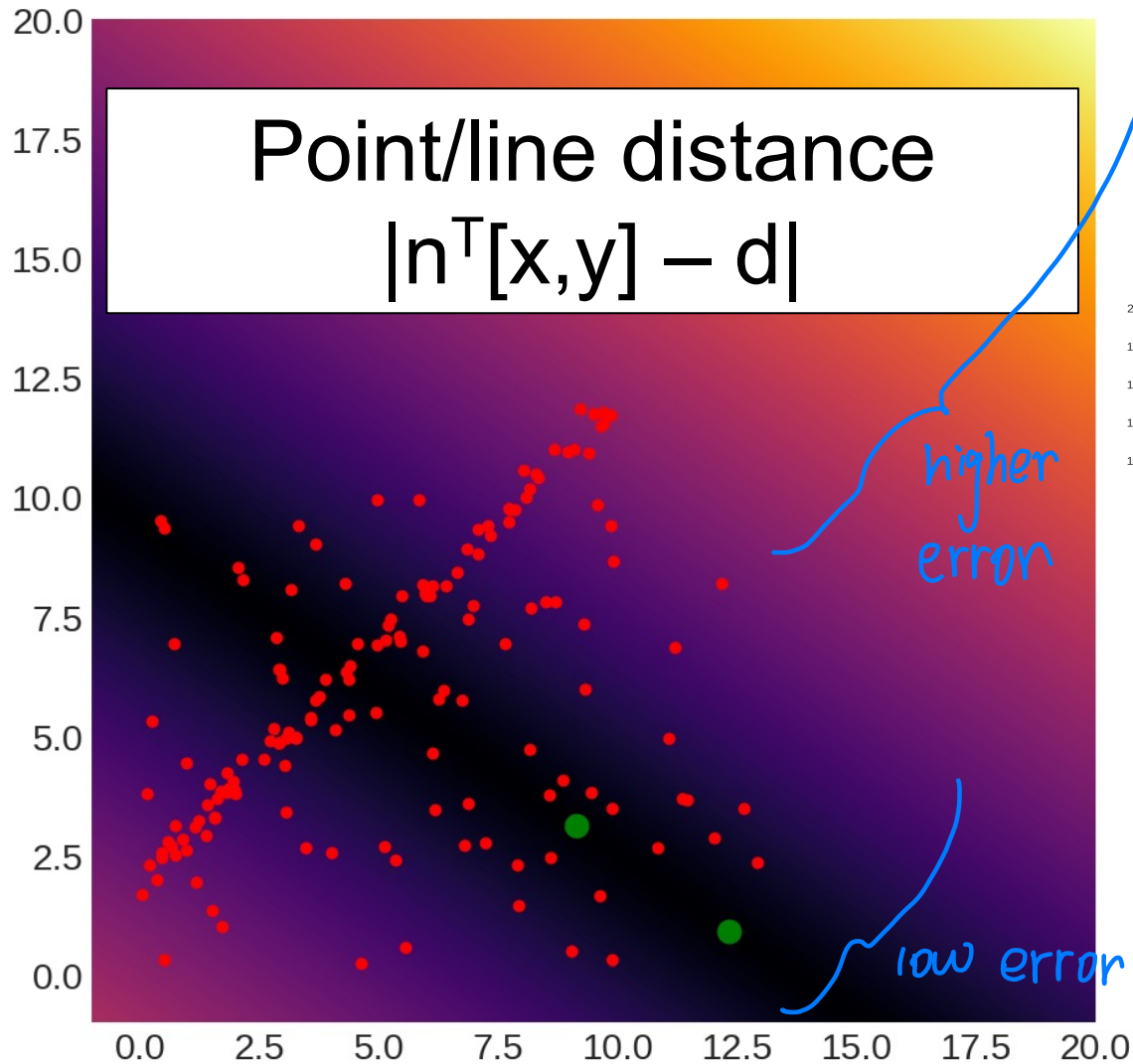
Best
Model:



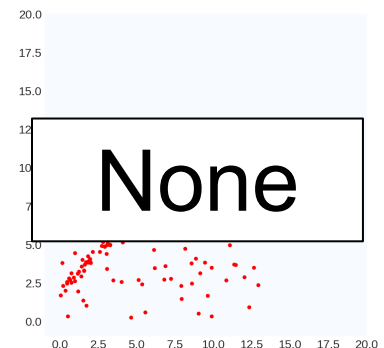
Best
Count:
-1

Running RANSAC

Trial
#1



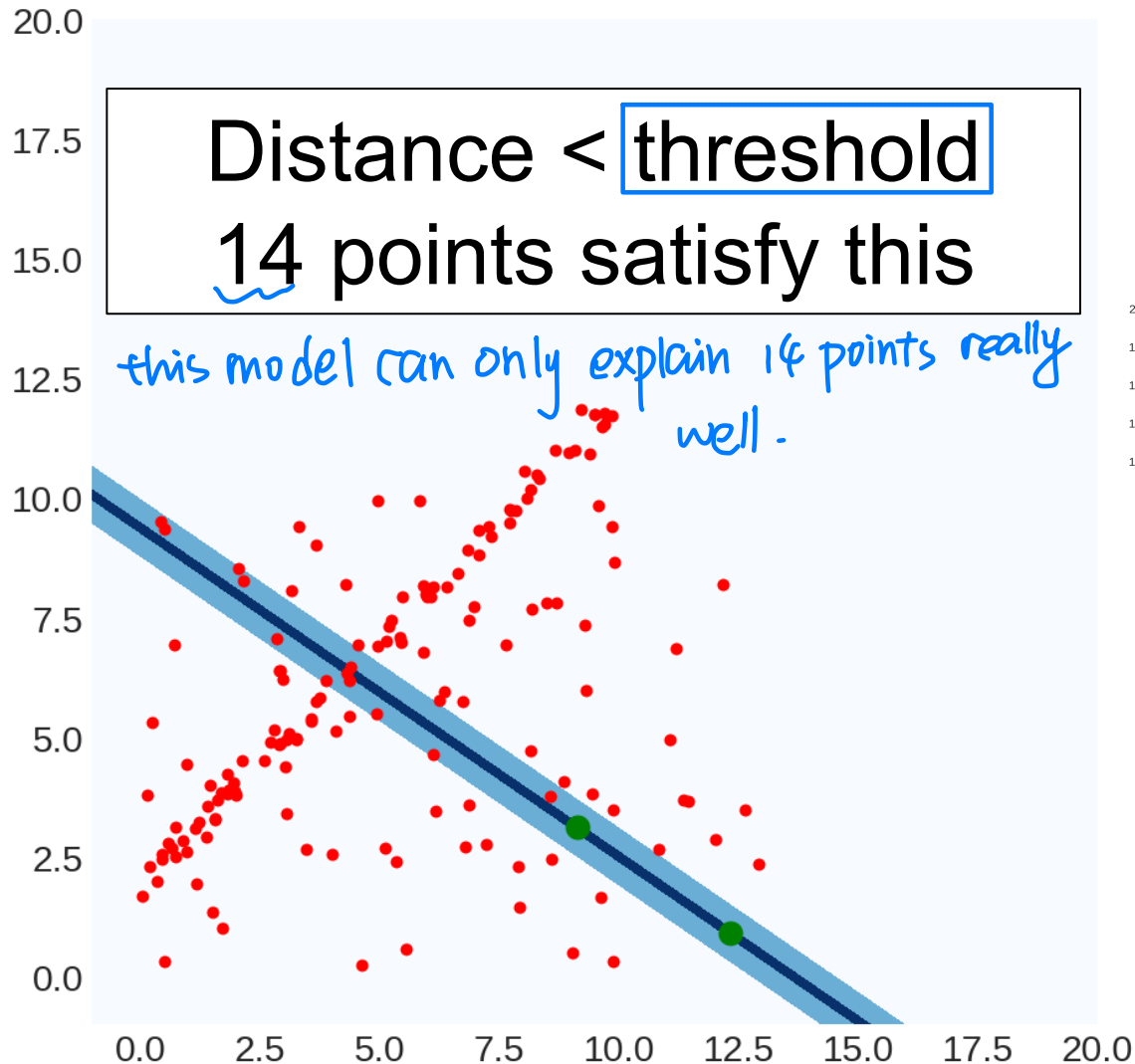
Best
Model:



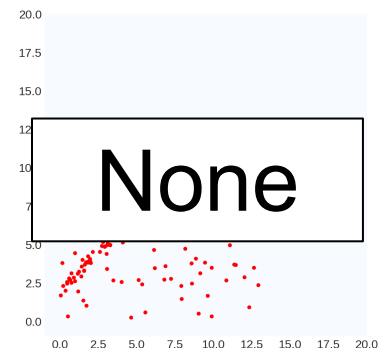
Best
Count:
-1

Running RANSAC

Trial
#1



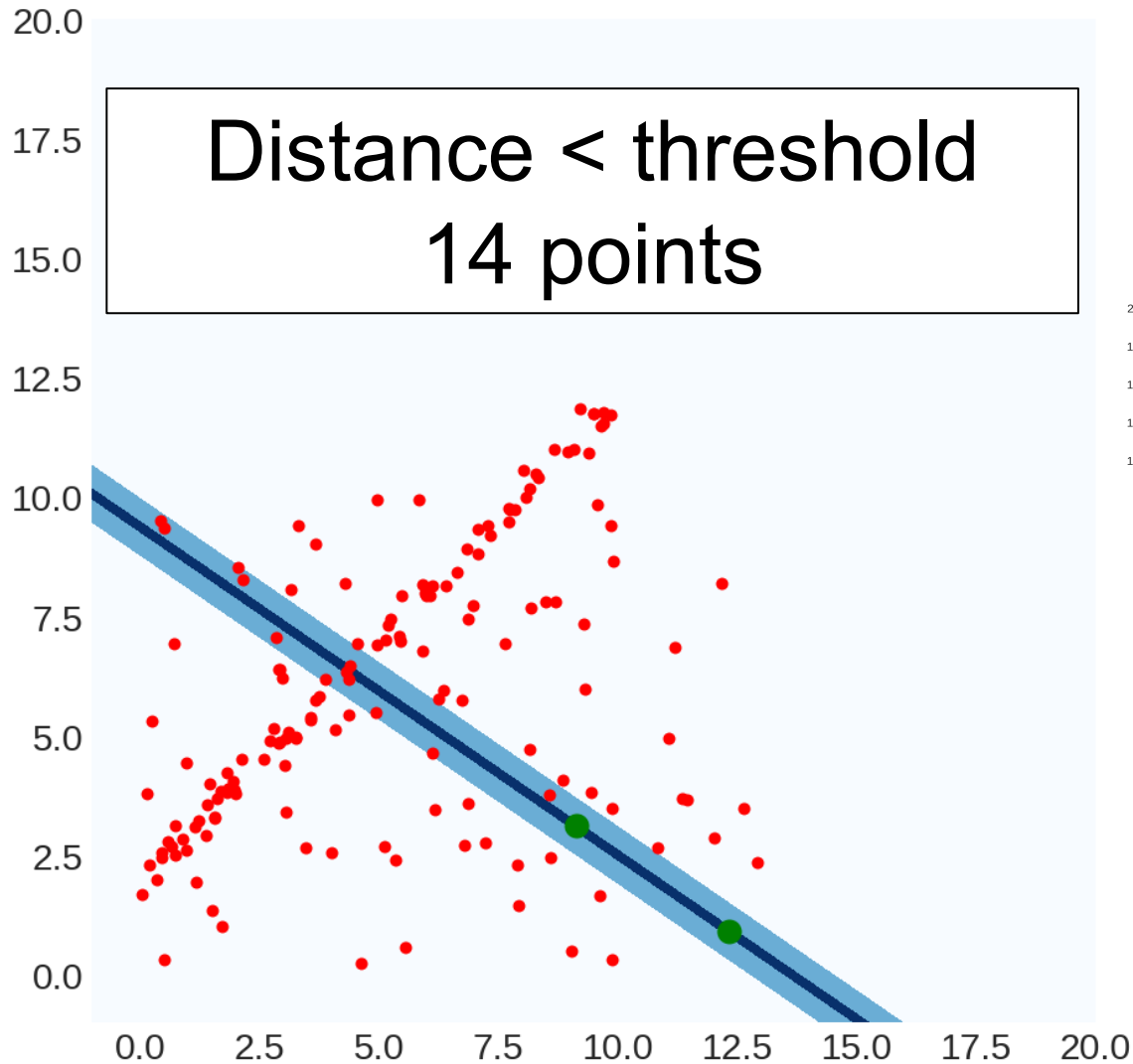
Best
Model:



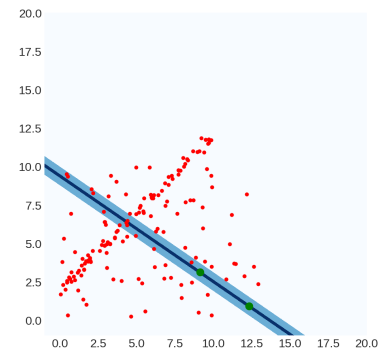
Best
Count:
-1

Running RANSAC

Trial
#1



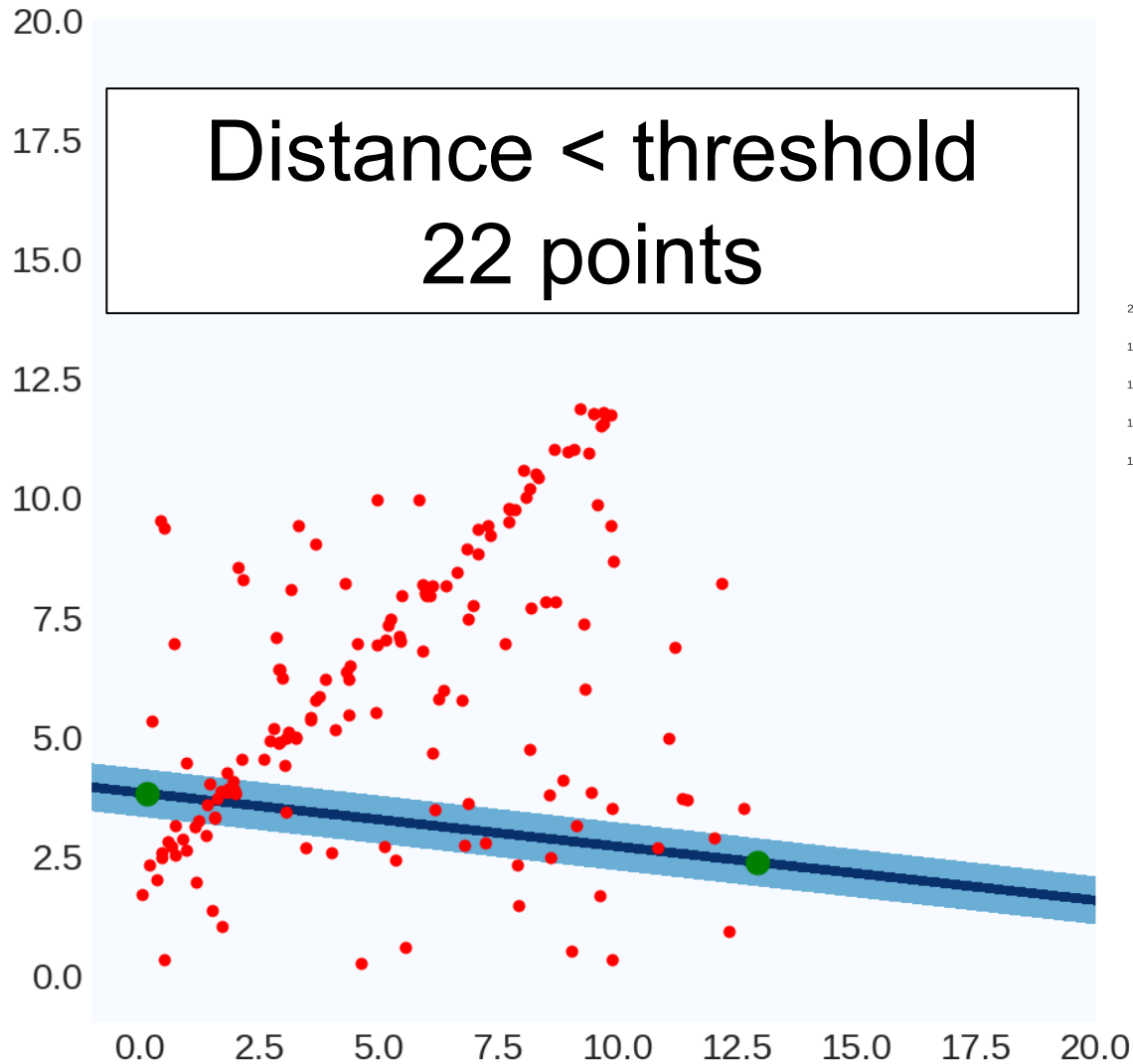
Best
Model:



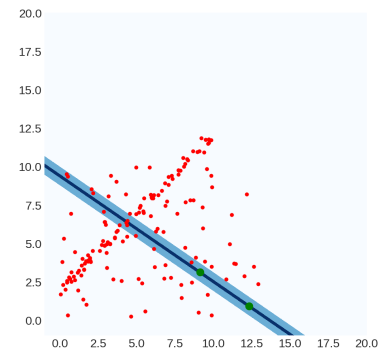
Best
Count:
14

Running RANSAC

Trial
#2



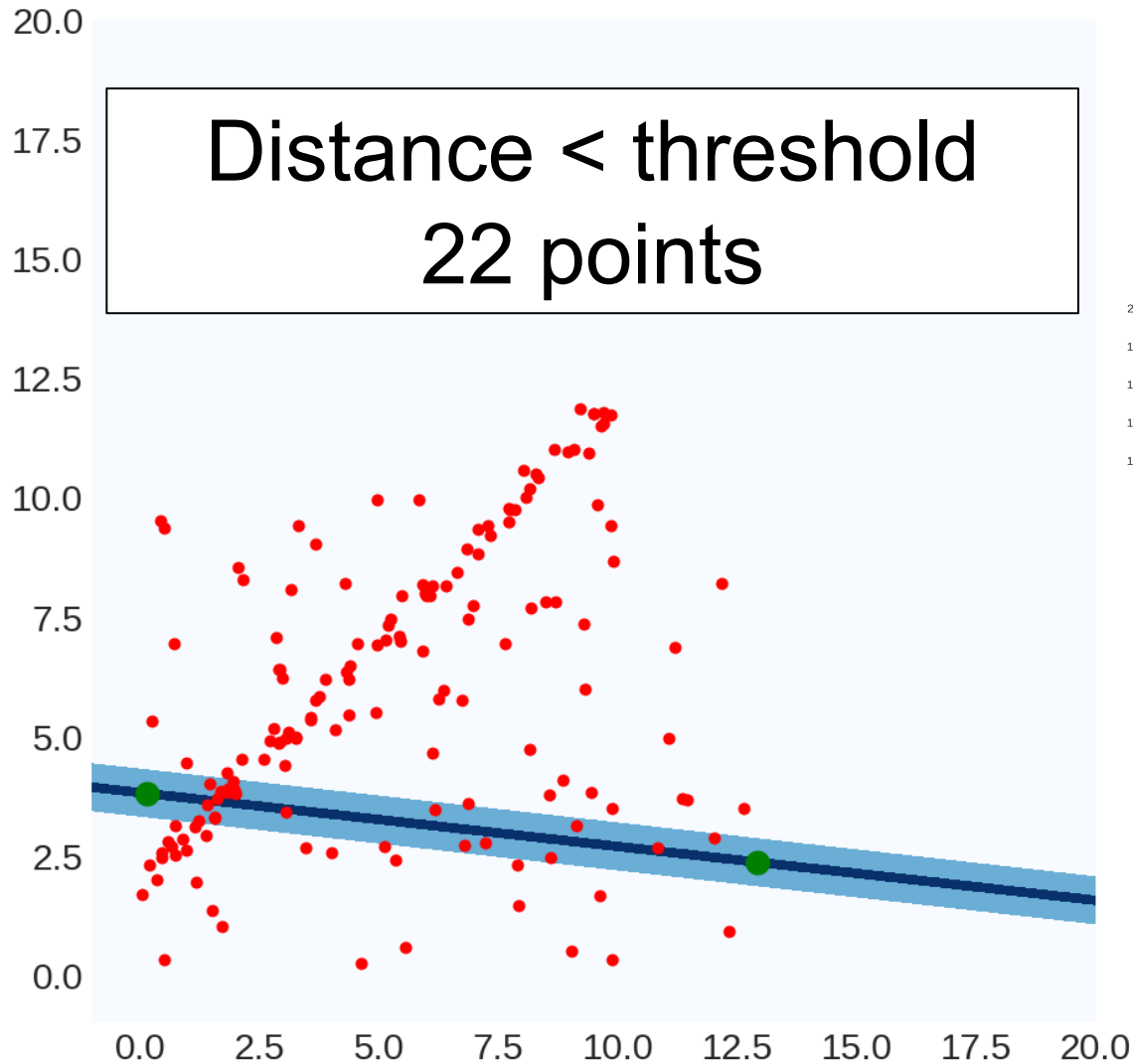
Best
Model:



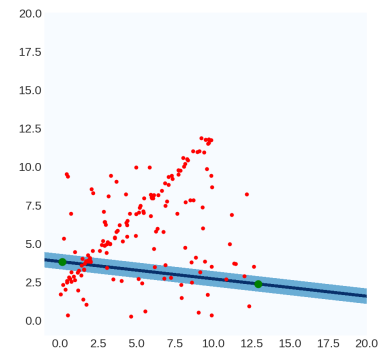
Best
Count:
14

Running RANSAC

Trial
#2



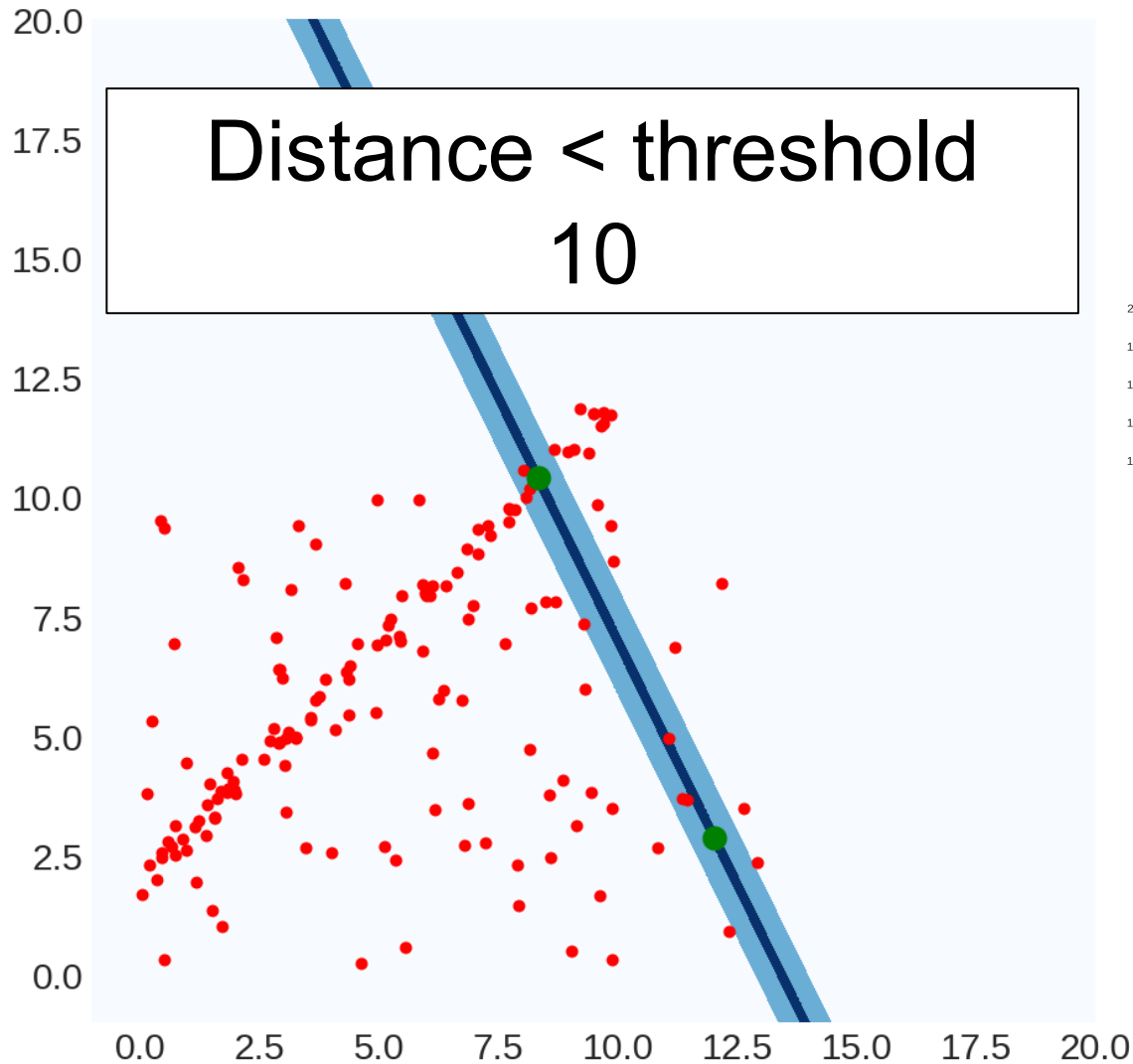
Best
Model:



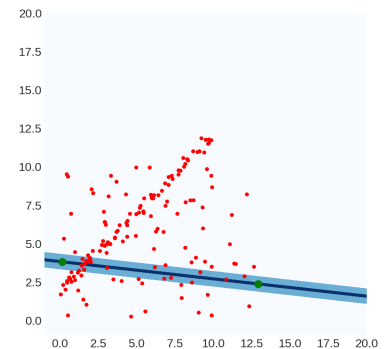
Best
Count:
22

Running RANSAC

Trial
#3



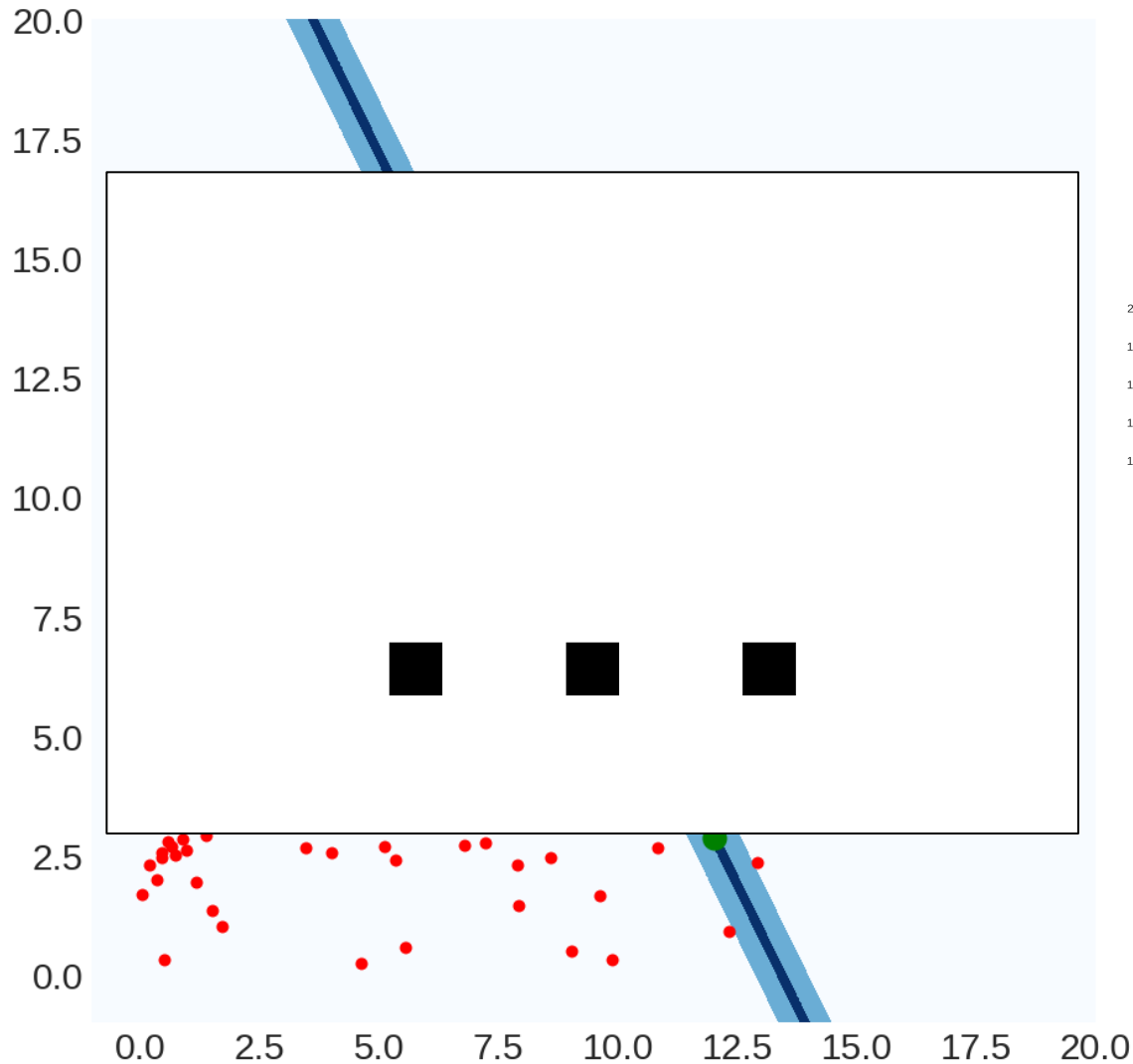
Best
Model:



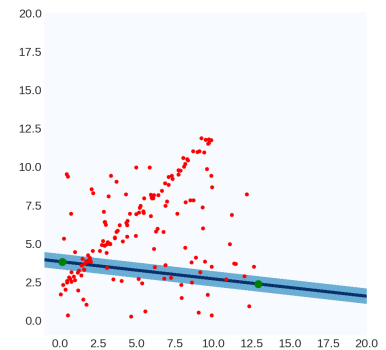
Best
Count:
22

Running RANSAC

Trial
#3



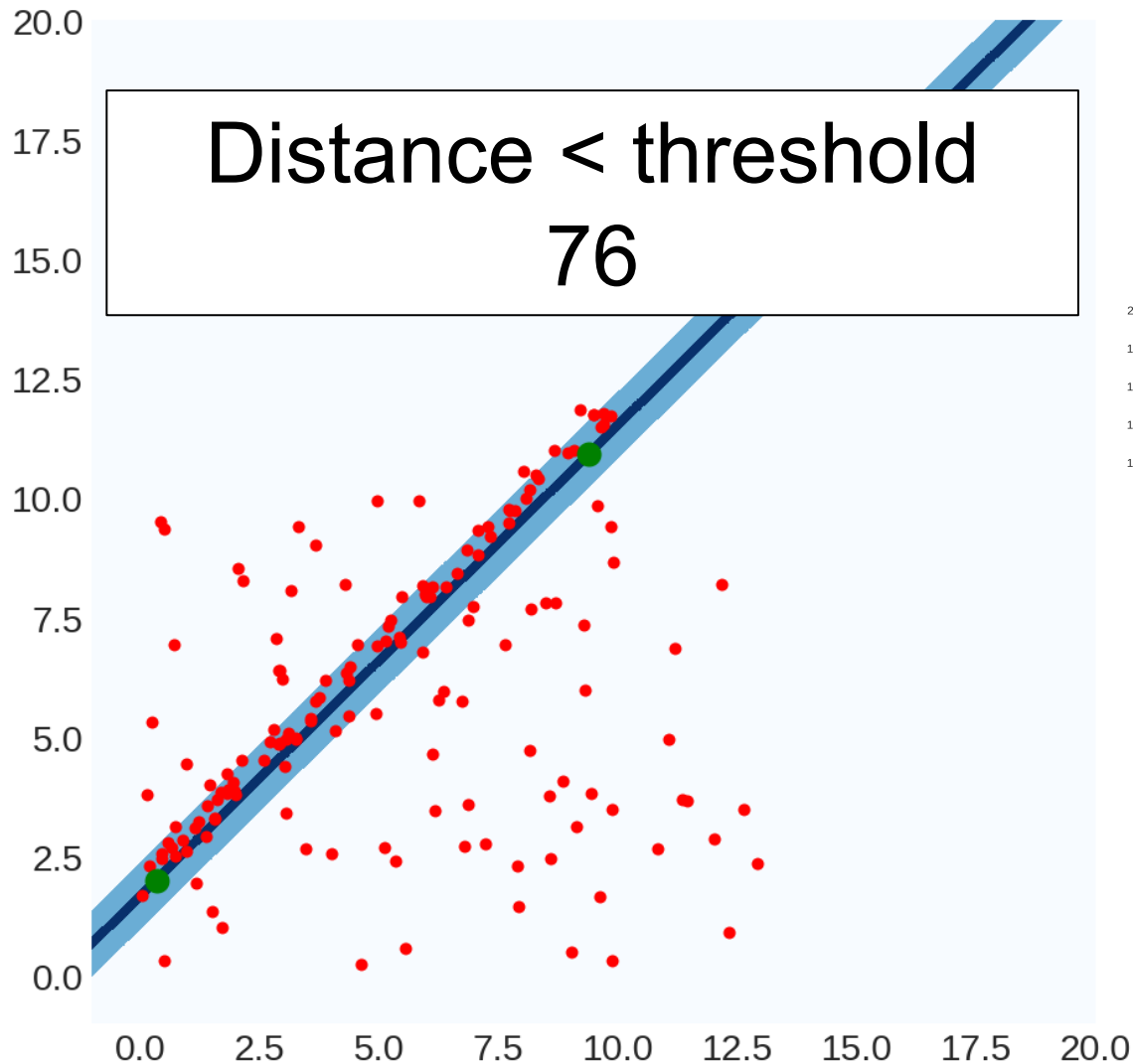
Best
Model:



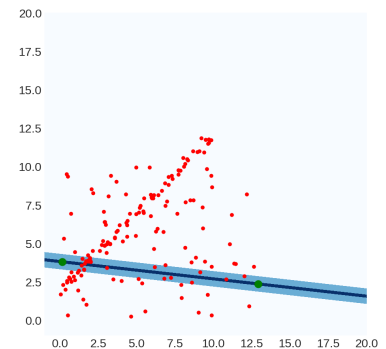
Best
Count:
22

Running RANSAC

Trial
#9



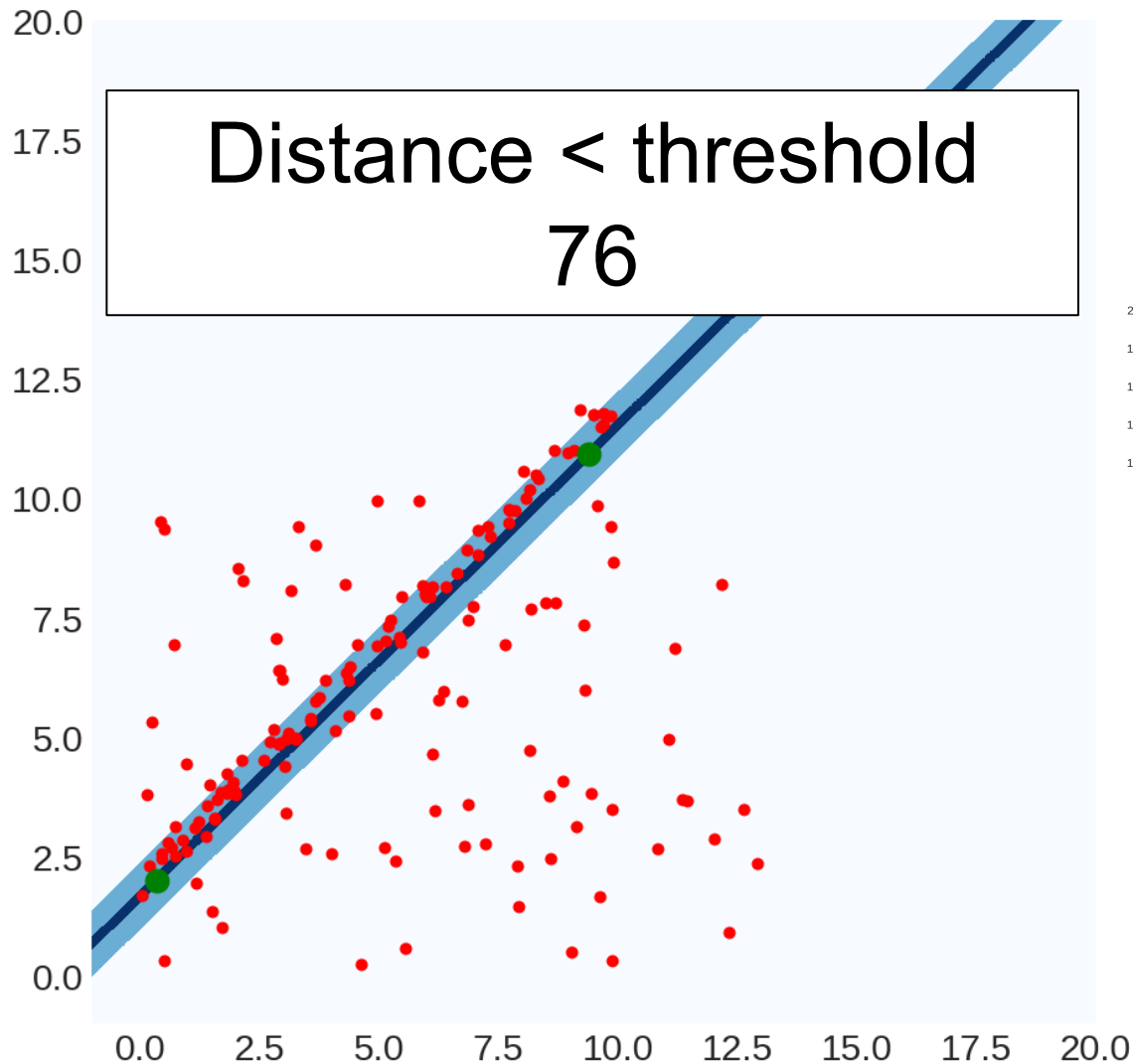
Best
Model:



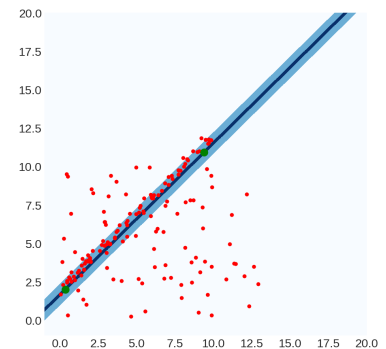
Best
Count:
22

Running RANSAC

Trial
#9



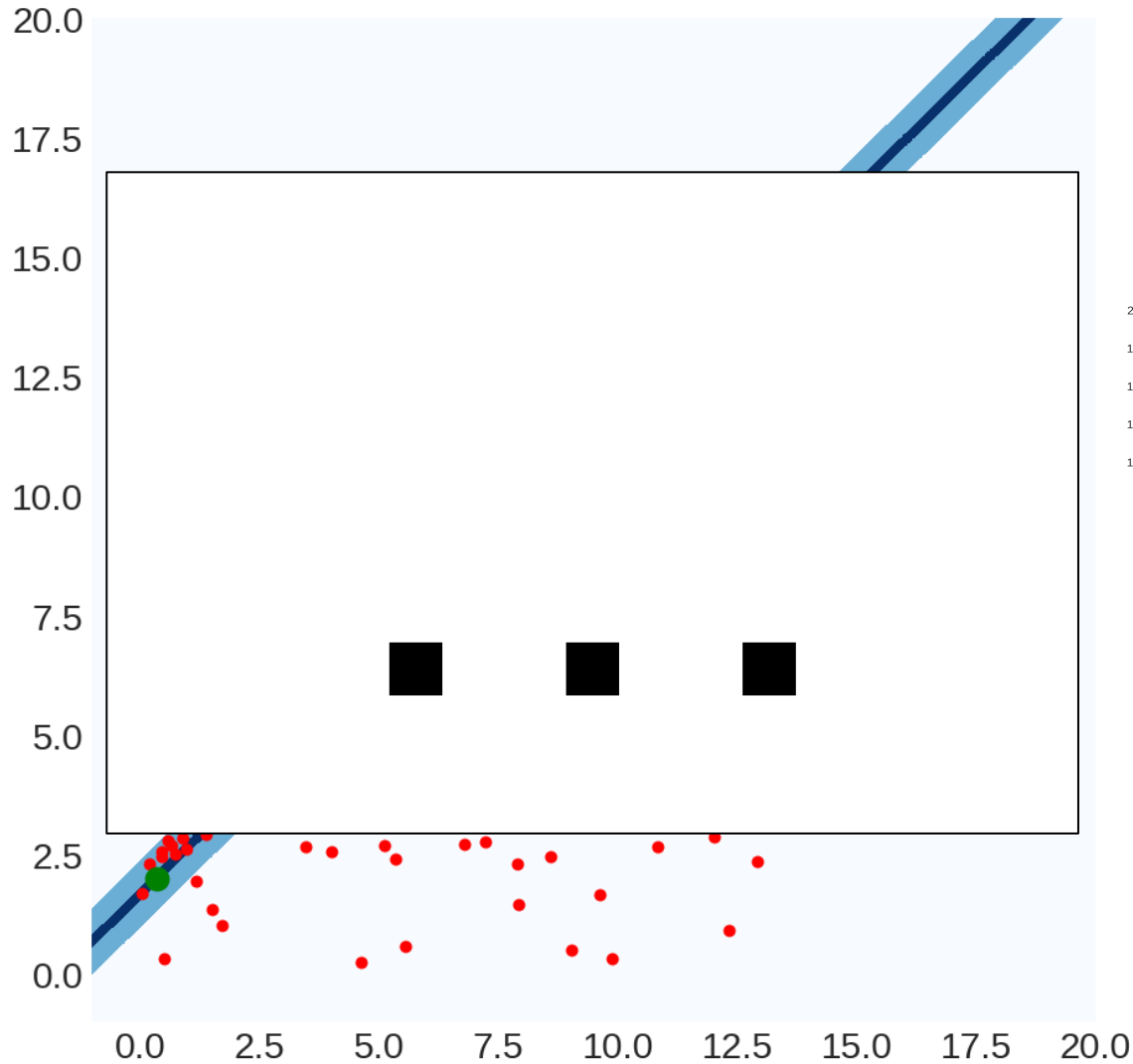
Best
Model:



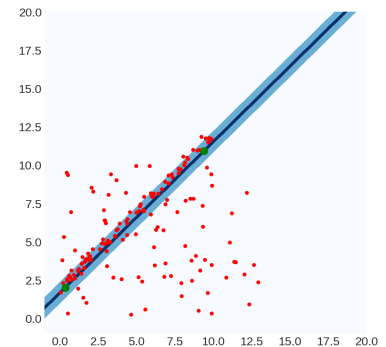
Best
Count:
76

Running RANSAC

Trial
#9



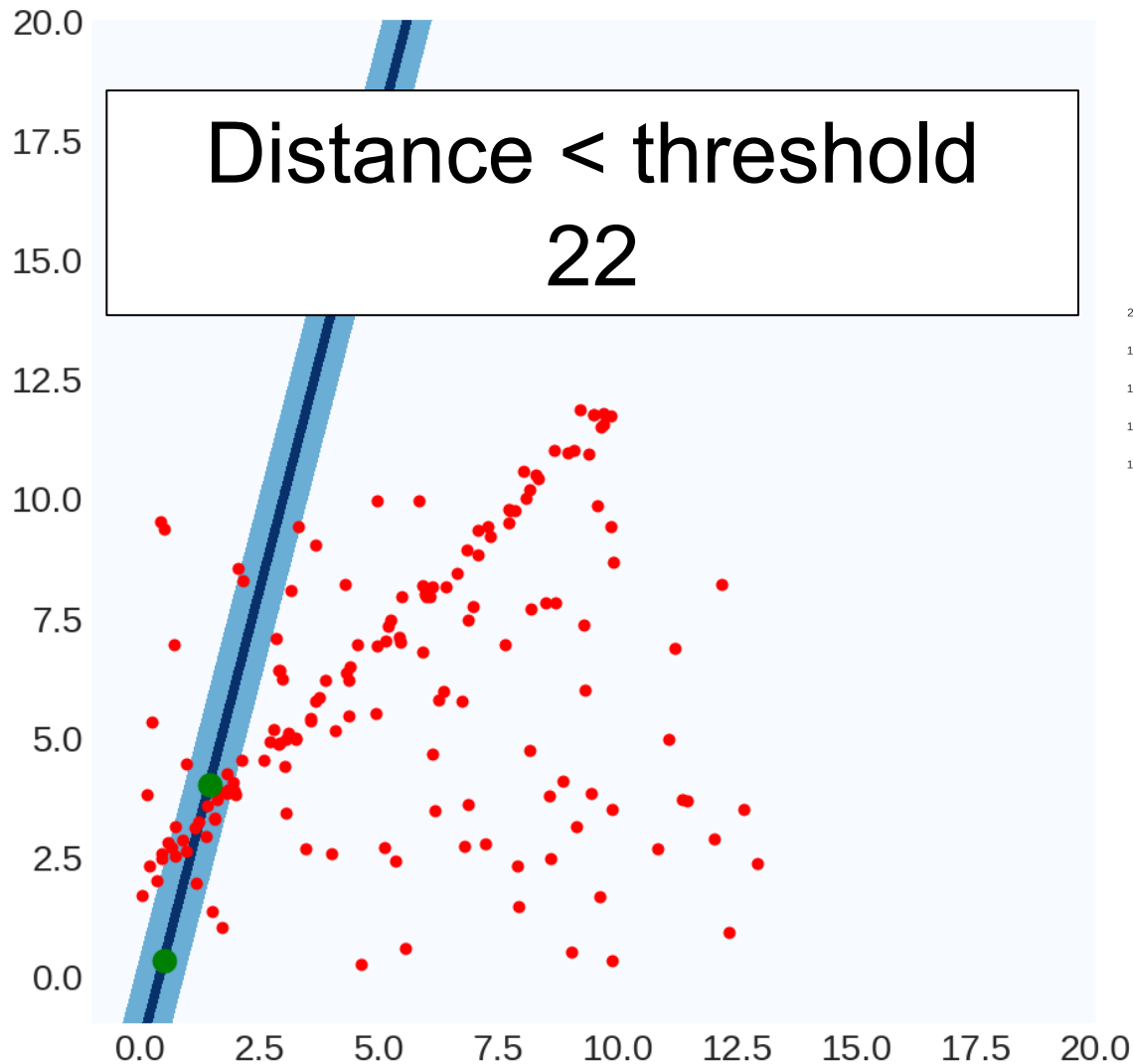
Best
Model:



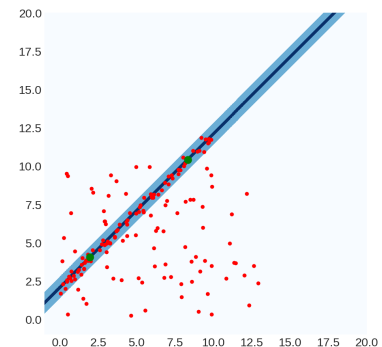
Best
Count:
76

Running RANSAC

Trial
#100

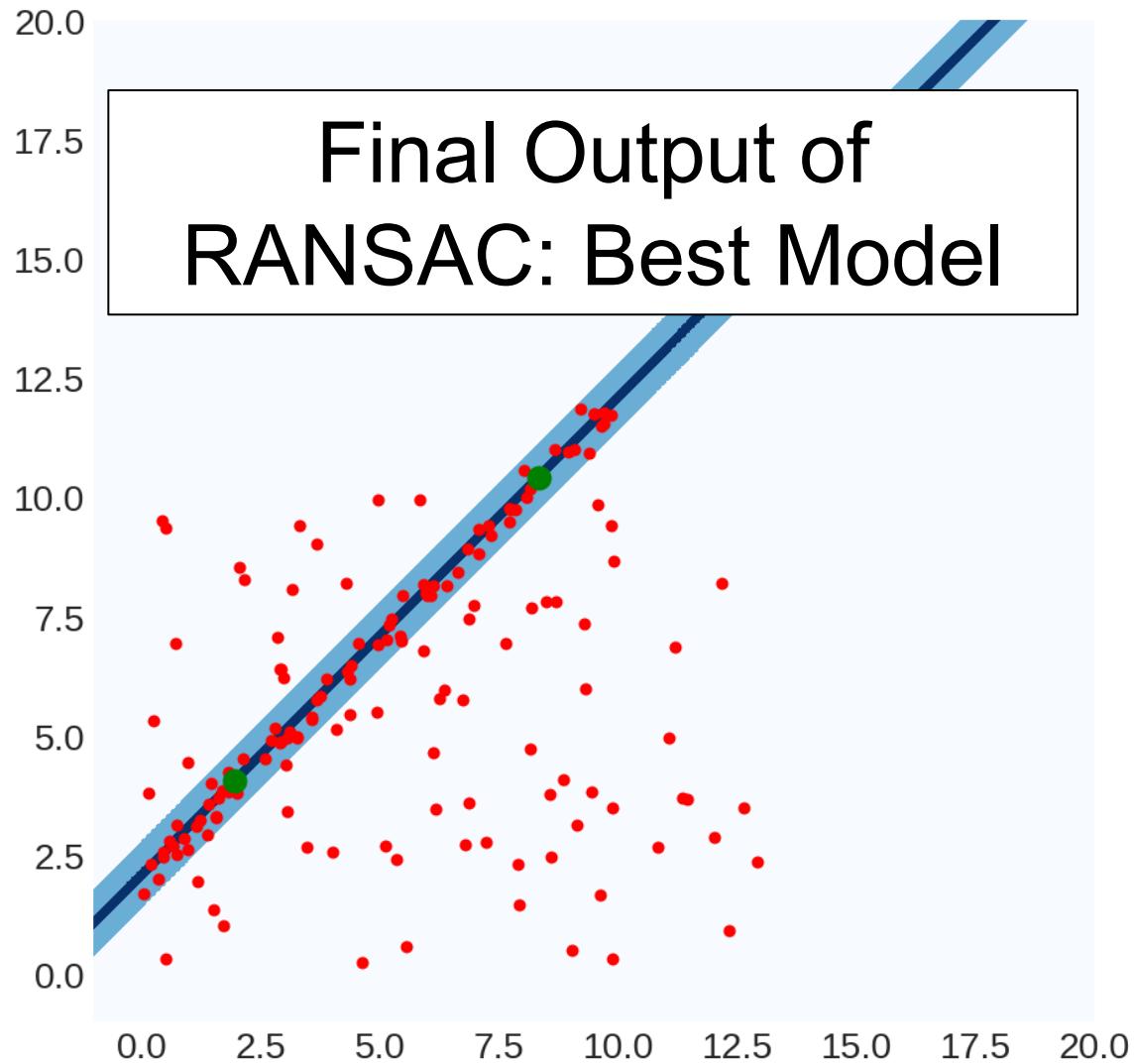


Best
Model:



Best
Count:
85

Running RANSAC



RANSAC In General

parameters we need to choose.

best, bestCount = None, -1

for trial in range(**NUM_TRIALS**):

① How many time we need to choose.

subset = pickSubset(data, **SUBSET_SIZE**)

②

model = fitModel(subset)

create the line based on how many points.

E = computeError(data, line)

inliers = E < **THRESHOLD**

③

if #(inliers) > bestCount:

best, bestCount = model, #(inliers)

Refit on the inliers for best model (keep these!)

Parameters – Num Trials

80% of the data is unreliable

r is the fraction of outliers (e.g., 80%)

Suppose we pick s points (e.g., 2) *subset size*
we run RANSAC N times (e.g., 500) *iteration time*

What's the probability of picking a sample set with no outliers?

$$\approx (1 - r)^s$$

(4%) 两个点都是 inlier 的概率

What's the probability of picking a sample set with any outliers?

$$1 - (1 - r)^s$$

(96%)

Parameters – Num Trials

r is the fraction of outliers (e.g., 80%)

Suppose we pick s points (e.g., 2)

we run RANSAC N times (e.g., 500)

What's the probability of picking a sample set with any outliers?

$$1 - (1 - r)^s \quad \textbf{(96\%)}$$

What's the probability of picking only sample sets with outliers?

$$(1 - (1 - r)^s)^N \quad \textbf{(10^{-7}\% N=500)}$$

$$\textbf{(13\% N=50)}$$

What's the probability of picking at least one set with inliers?

$$1 - (1 - (1 - r)^s)^N \quad (\sim 100\% \quad N=500)$$

$$(87\% \quad N=50)$$

Parameters – Num Trials



$P(\text{Jackpot}):$
 $1 / 302,575,350$

Death by
vending
machine



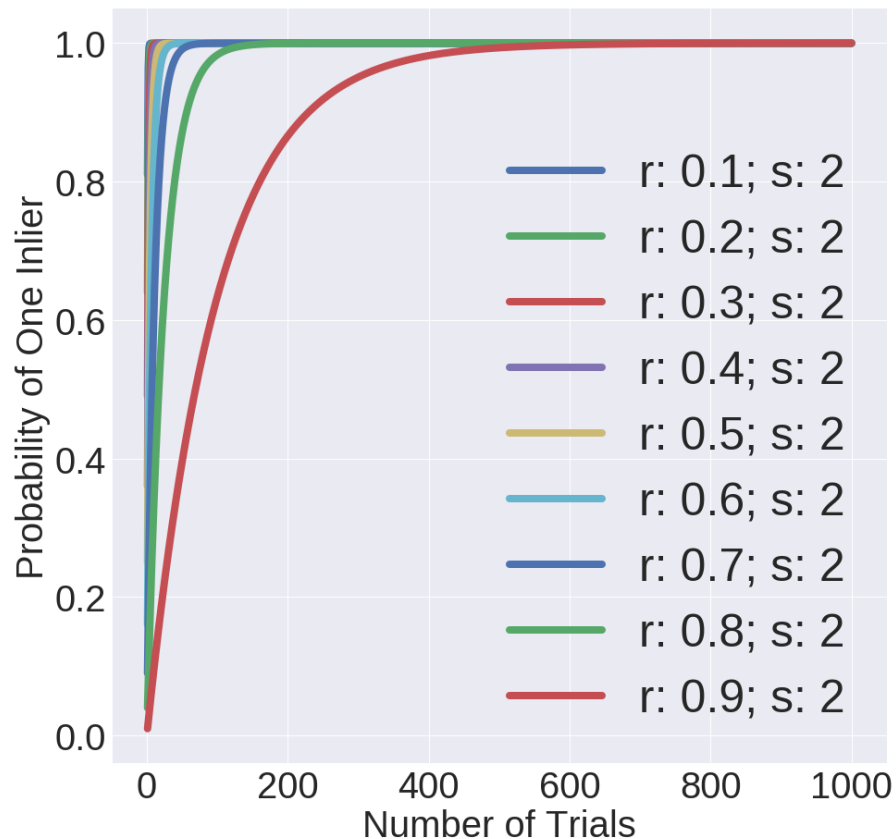
$P(\text{Death}):$
 $\approx 1 / 112,000,000$

RANSAC fails to fit a
line with 80% outliers
after trying only 500
times

$P(\text{Failure}):$
 $1 / 731,784,961$

Parameters – Num Trials

r is the fraction of outliers (e.g., 80%)
Suppose we pick s points (e.g., 2)
we run RANSAC N times (e.g., 500)



Parameters – Subset Size

- Always the smallest possible set for fitting the model.
- Minimum number for lines: 2 data points
- Minimum number for 3D planes: **how many?** 3

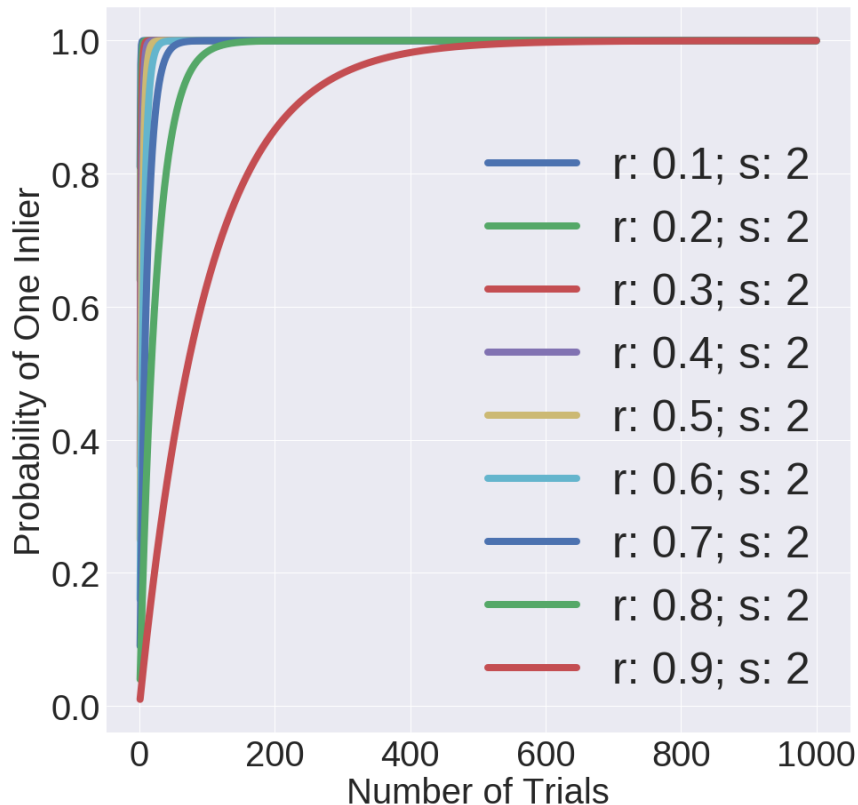
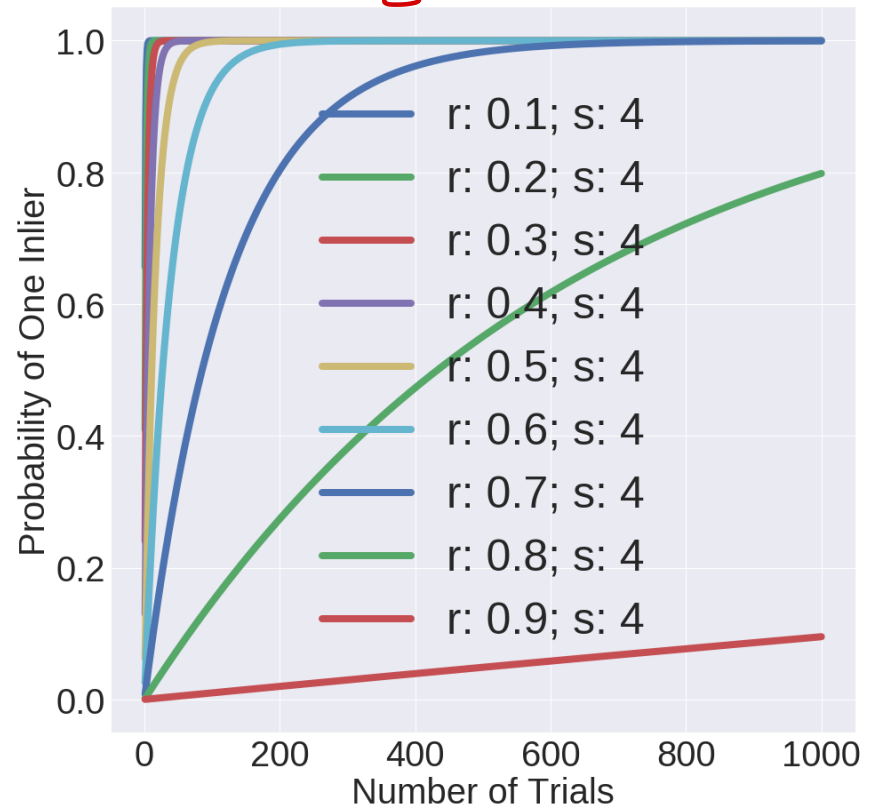
- **Why the minimum intuitively?** *A: the cost of iteration. Also, as you have larger subset size, the chance of at least one outlier hiding inside this subset gets larger*
- **Probability failure increases exponentially with higher s !**
- You'll find out more in homework 3.

Parameters – Num Trials

r is the fraction of outliers (e.g., 80%)

Suppose we pick s points (e.g., 2)

we run RANSAC N times (e.g., 500)

$$S=2$$

$$S = 4$$


Parameters – Threshold

- No magical threshold

RANSAC Pros and Cons

Pros

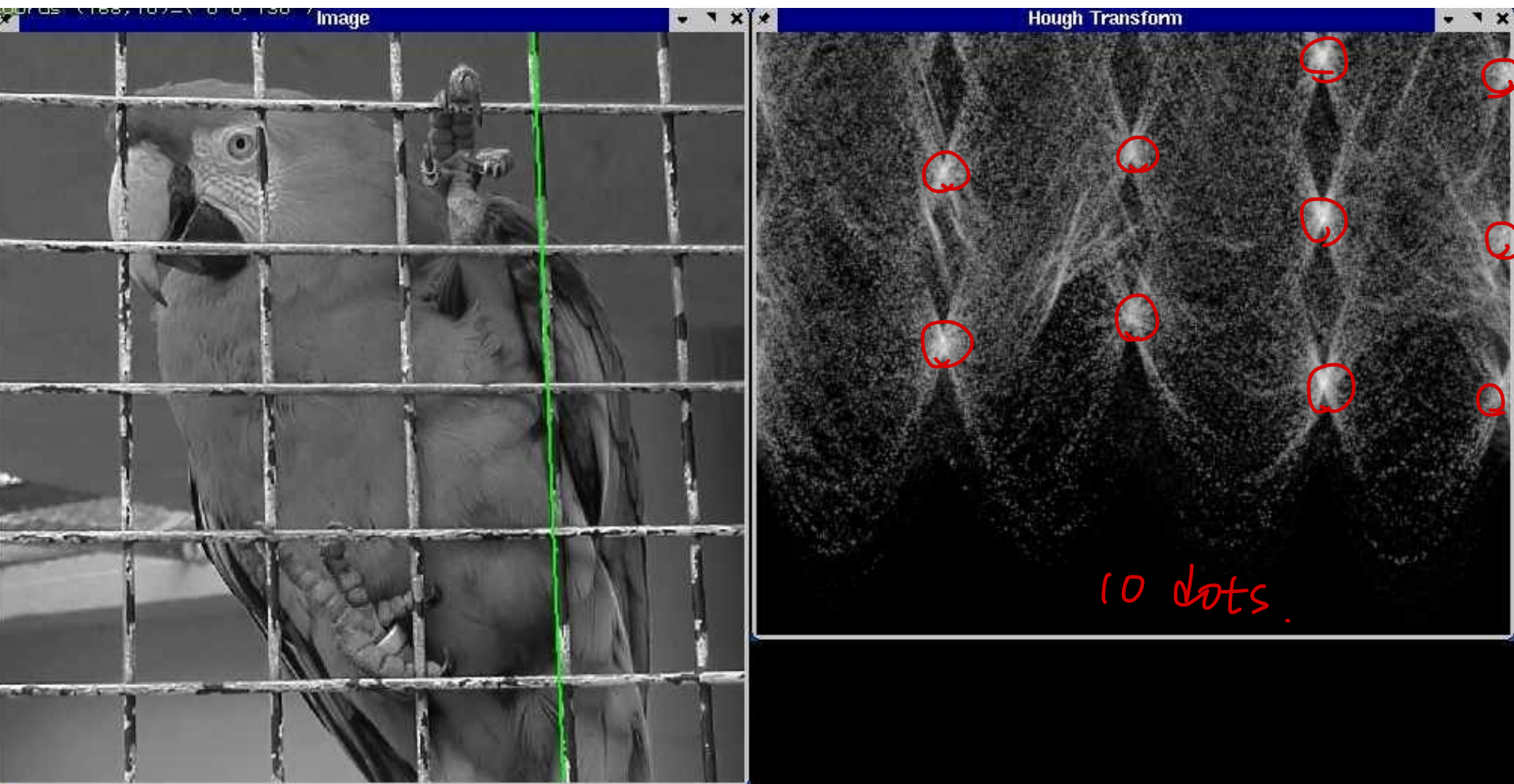
1. Ridiculously simple
2. Ridiculously effective
3. Works in general

Cons

1. Have to tune parameters *g: thresholding & iteration times.*
2. Weak theory (so can't derive parameters via theory)
3. Not magic, especially with lots of outliers

Finding lines effectively given outliers.

Alternative: Hough Transform



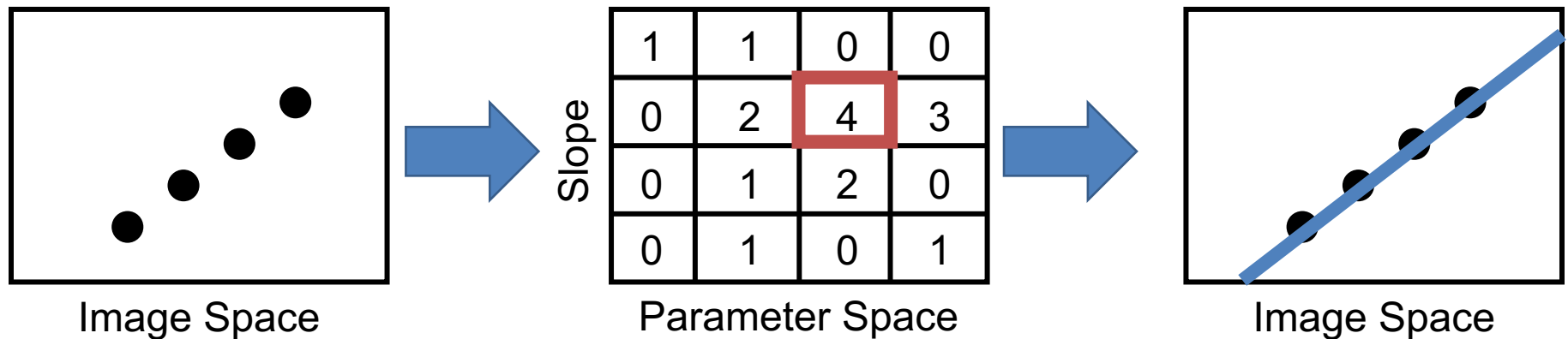
10 lines.

How many lines do you see?

For expressing a line, we have parameters \longrightarrow slope & intercept

Hough Transform

1. Discretize space of parametric models
2. Each pixel (data) votes for all compatible models
-typically for boundary (edge) pixels
3. Find models compatible with many pixels (data)

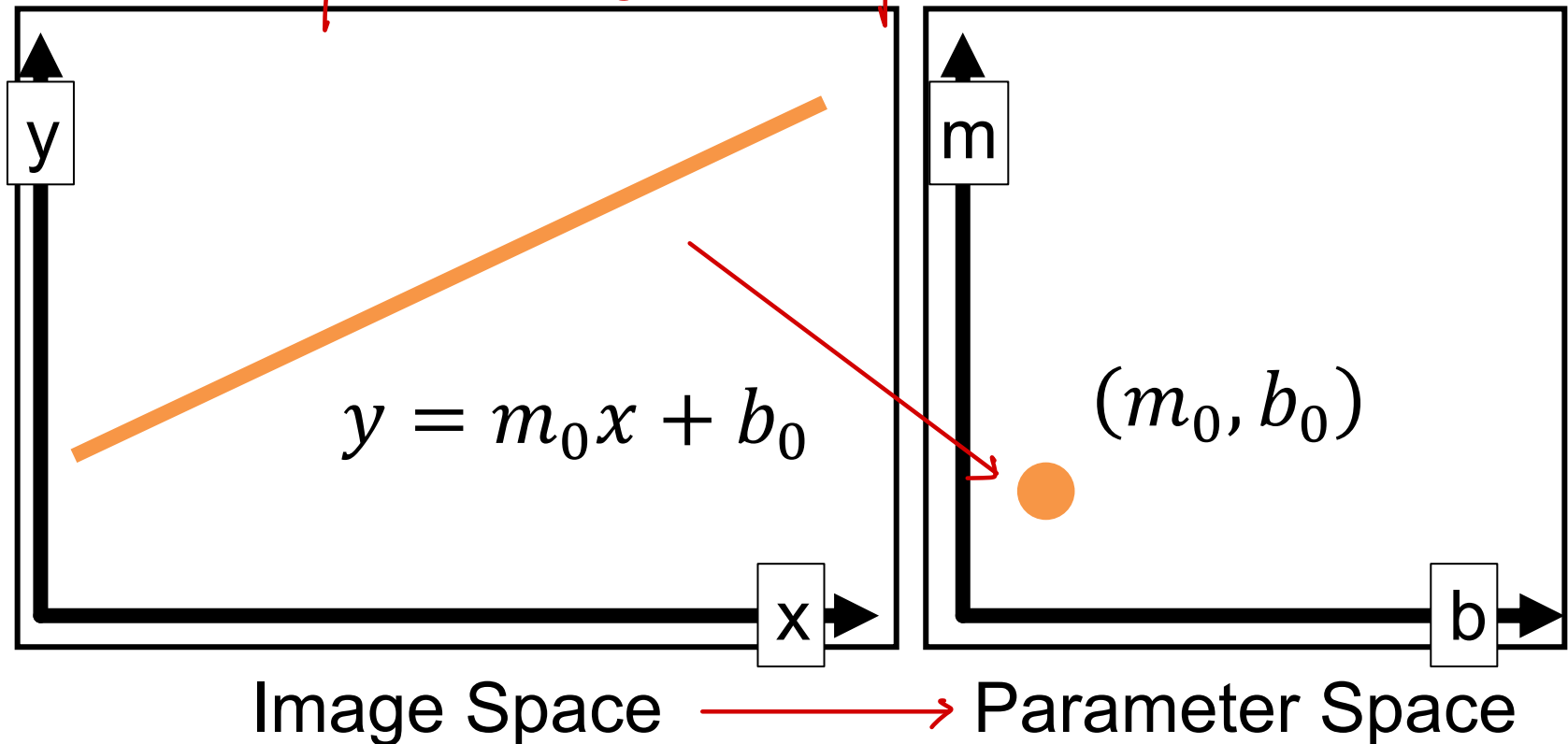


P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures*, Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

Hough Transform

Line in image = point in parameter space

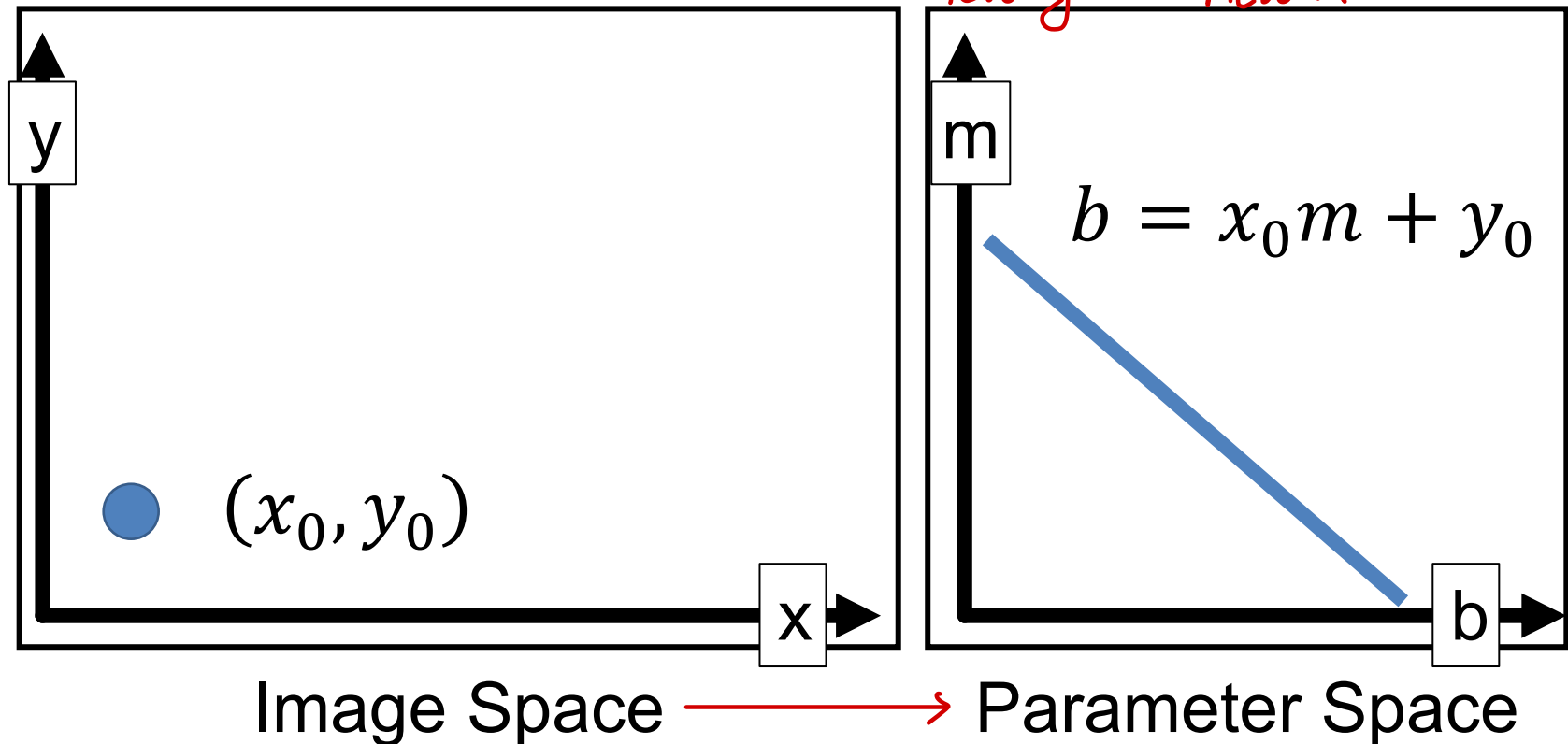
Transform a line into a point.



Hough Transform

Point in image = line in parameter space

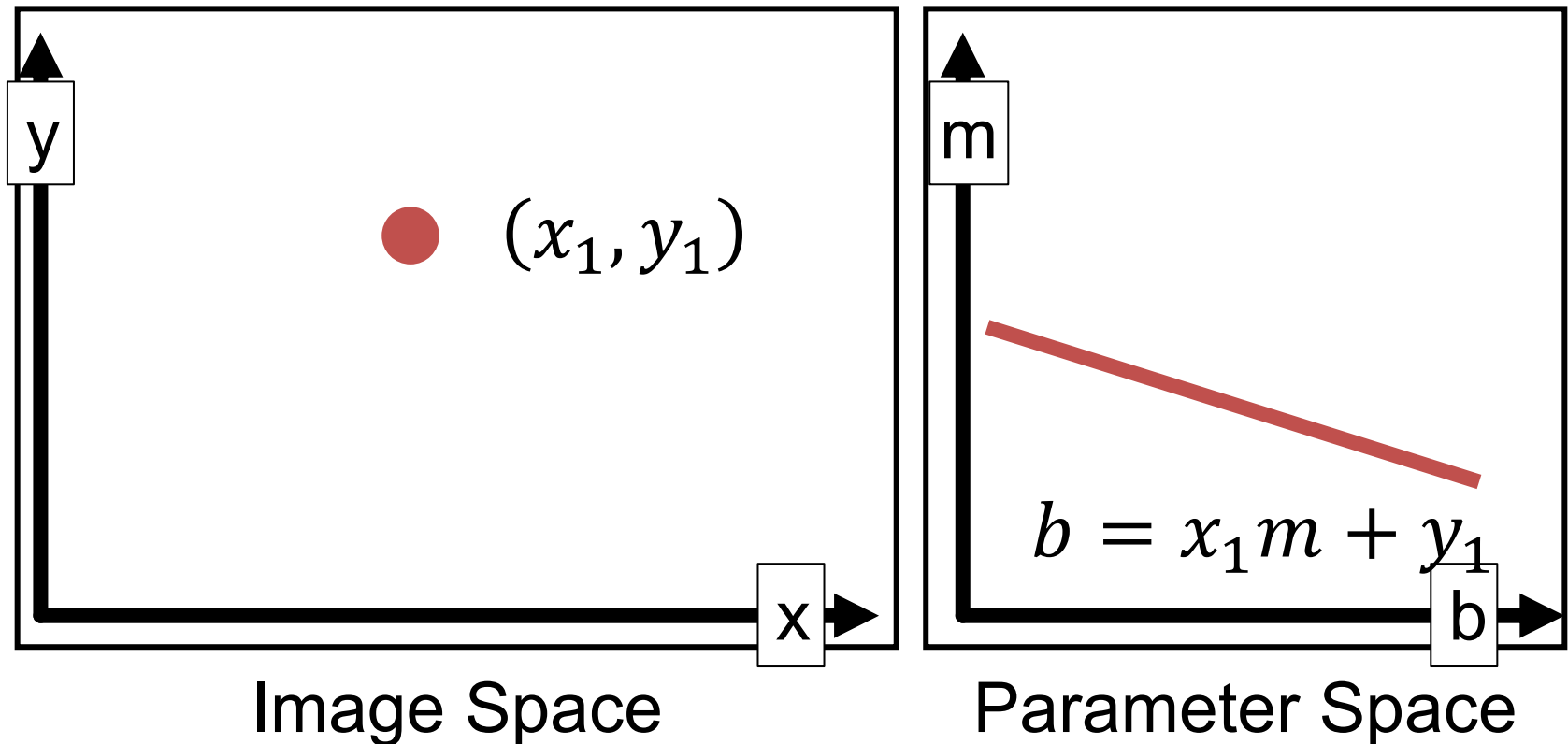
All lines through the point: $b = x_0 m + y_0$



Hough Transform

Point in image = line in parameter space

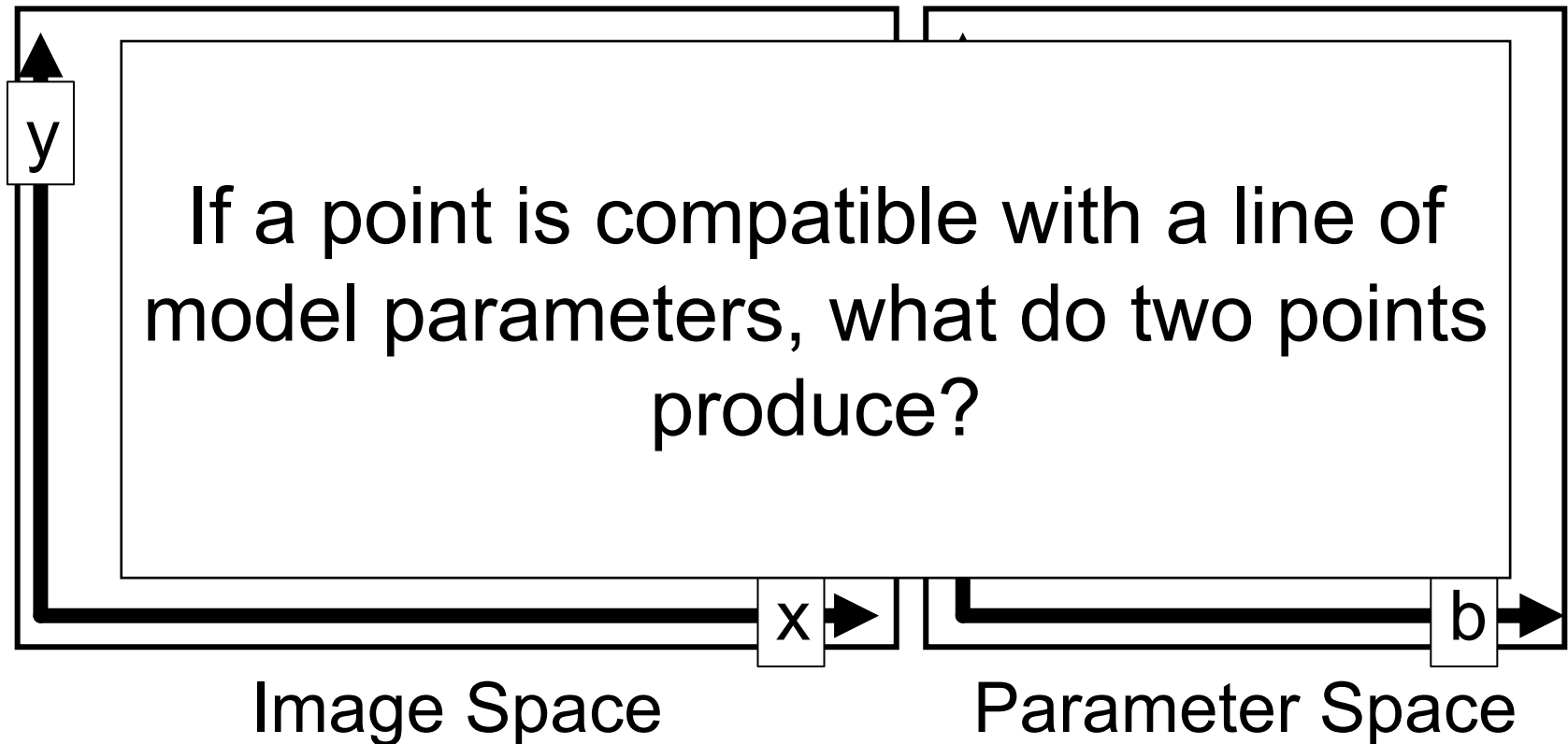
All lines through the point: $b = x_1 m + y_1$



Hough Transform

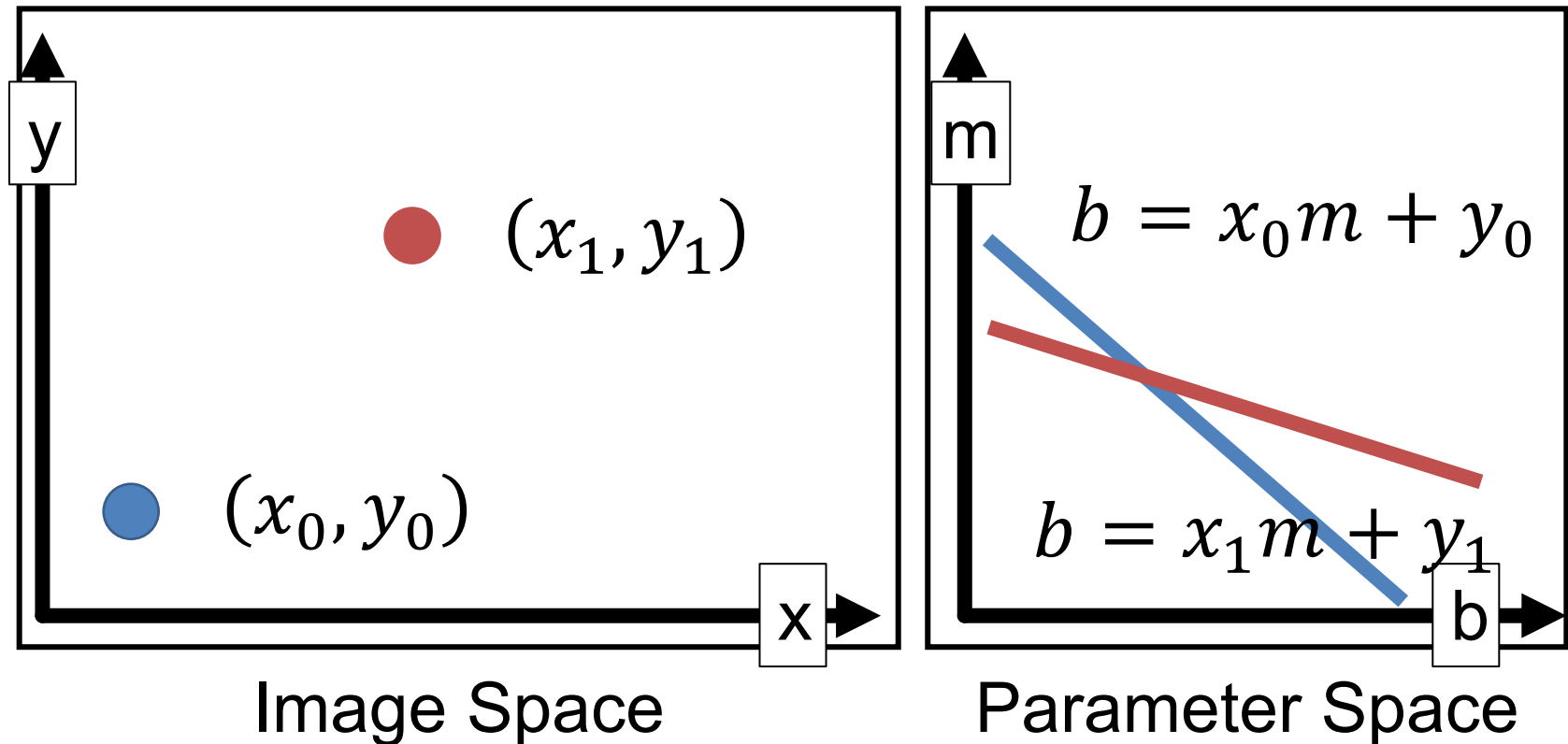
Point in image = line in parameter space

All lines through the point: $b = x_1 m + y_1$



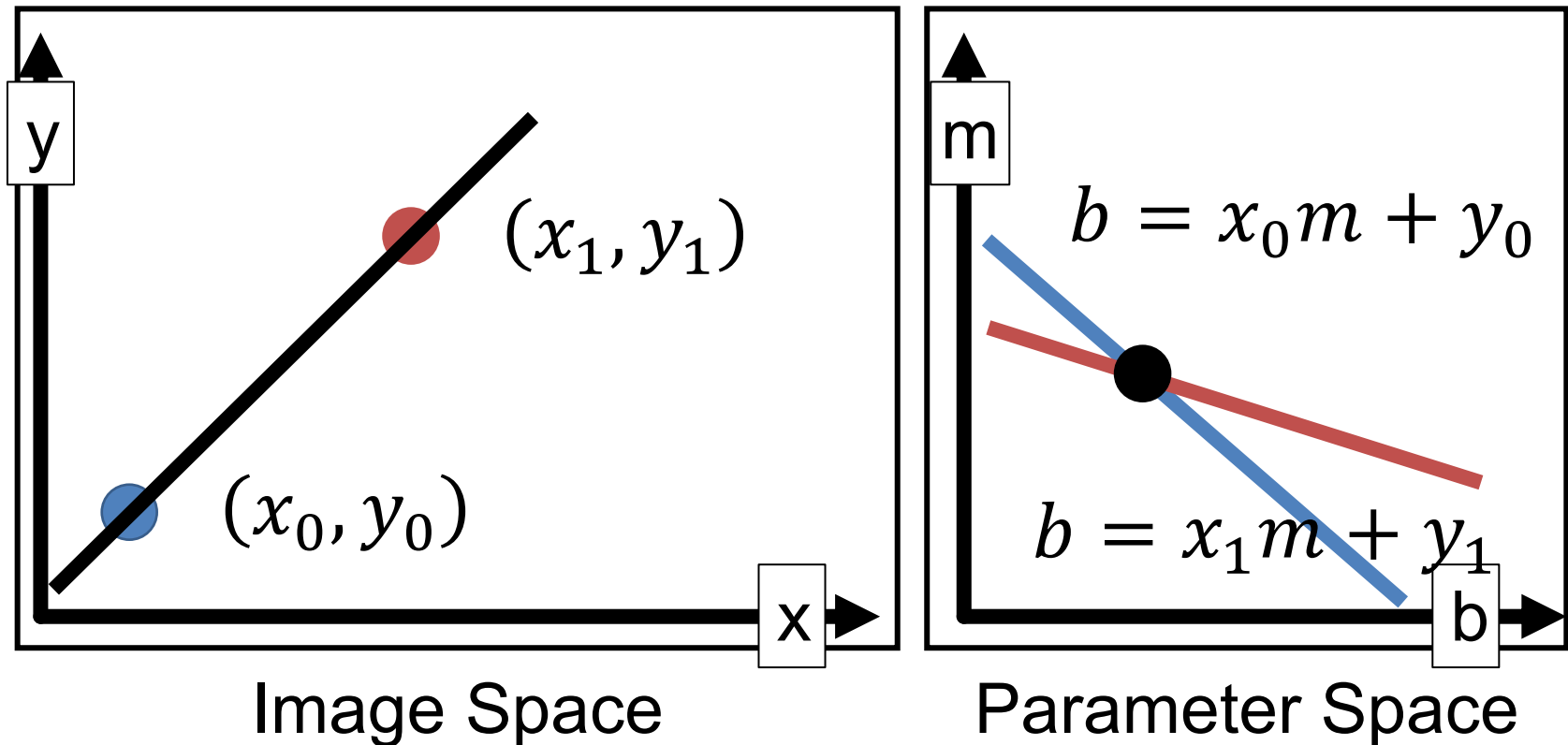
Hough Transform

Line through two points in image = intersection of two lines in parameter space (i.e., solutions to both equations)



Hough Transform

Line through two points in image = intersection of two lines in parameter space (i.e., solutions to both equations)



Hough Transform

- *Recall*: m, b space is awful
- $ax+by+c=0$ is better, but *unbounded*
- Trick: write lines using angle + offset (normally a mediocre way, but makes things bounded)

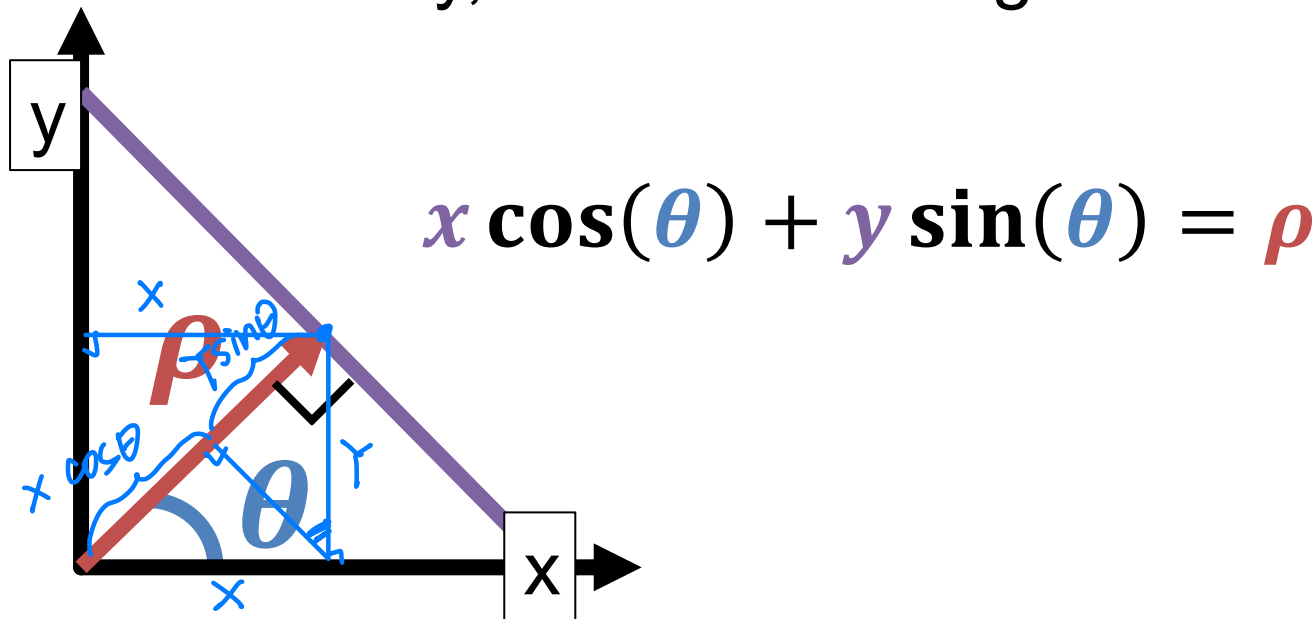


Diagram is remake of S. Seitz Slides; these are illustrative and values may not be real

Hough Transform Algorithm

Remember: $x \cos(\theta) + y \sin(\theta) = \rho$

parameters

Accumulator $H = \text{zeros}(\text{res}, \text{res})$

For x, y in points:

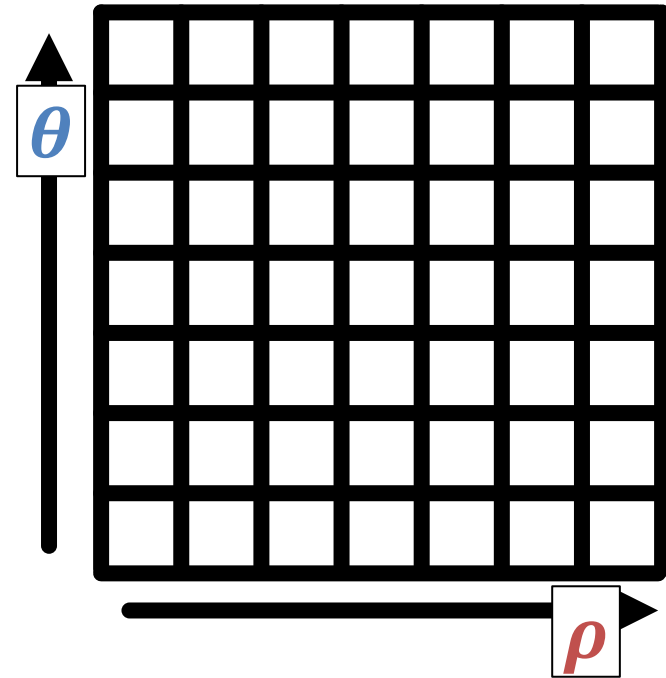
For θ in $\text{range}(0, 180, \text{res})$:

$$\rho = x \cos(\theta) + y \sin(\theta)$$

$$H[\theta, \rho] += 1$$

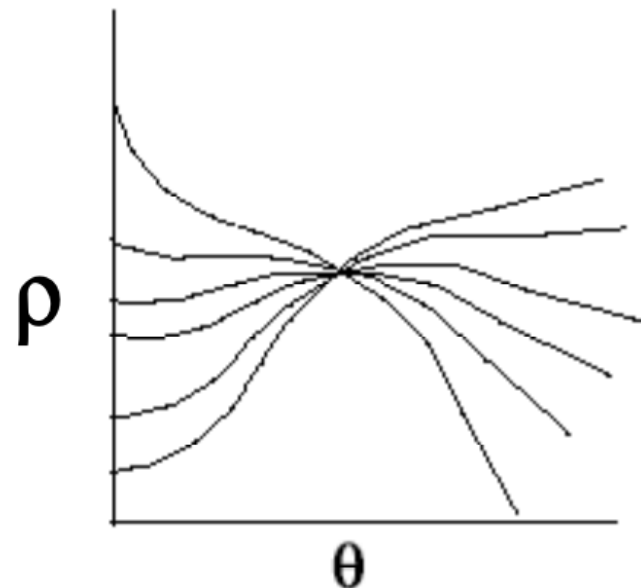
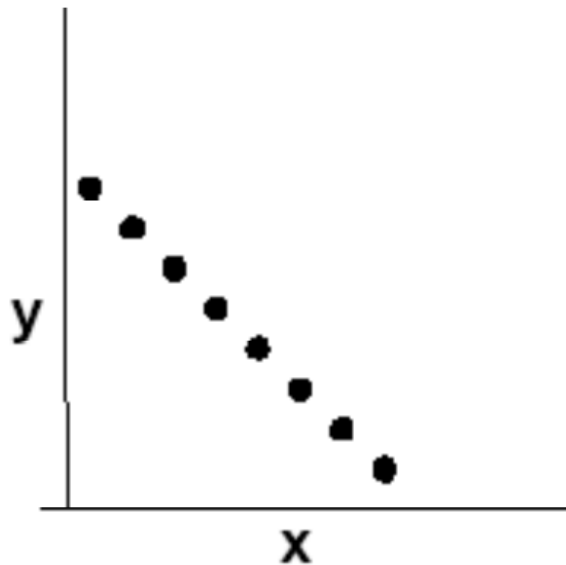
#any local maxima (θ, ρ) of H is a line

#of the form $\rho = x \cos(\theta) + y \sin(\theta)$



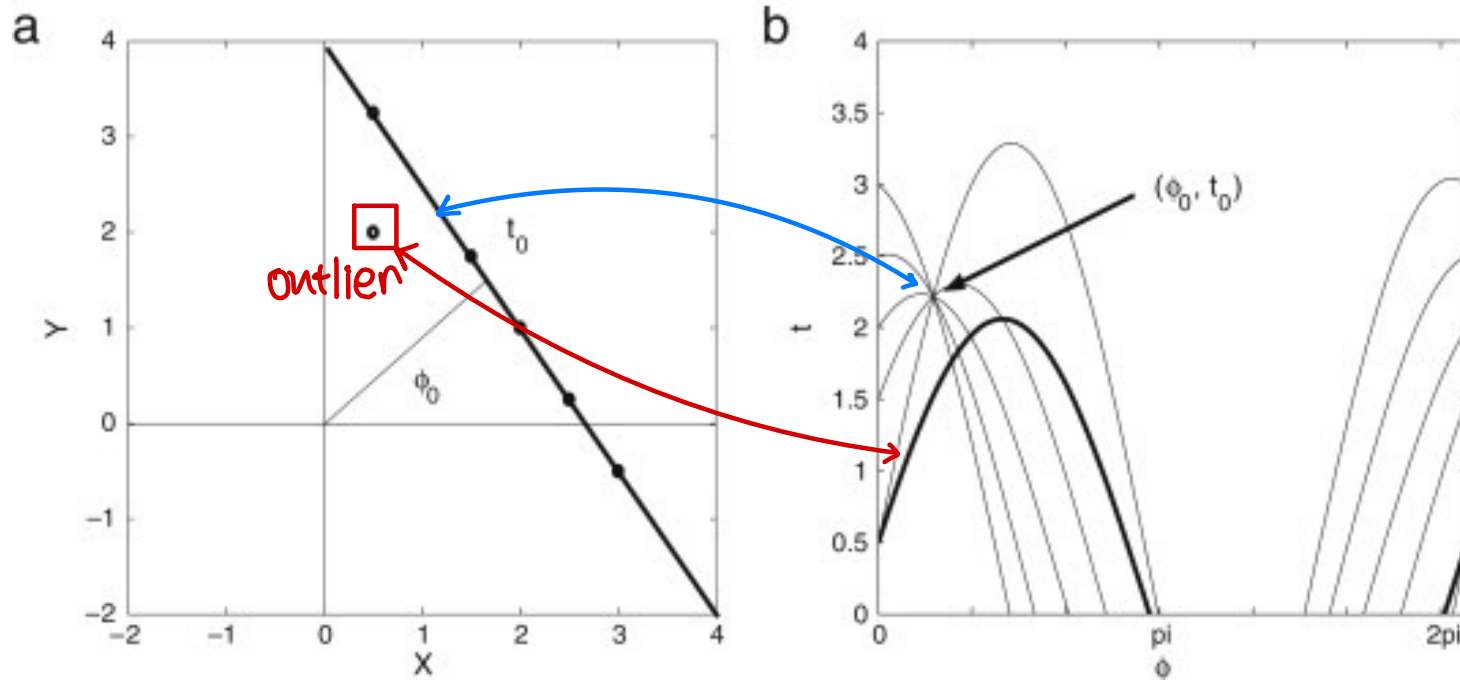
Coincidental Curves

- Each x, y point is a sinusoidal curve in the parameter space
- Points on a line coincide at a point



Coincidental Curves

- Each x, y point is a sinusoidal curve in the parameter space
- Points on a line coincide at a point
- Effectively ignore outliers!



Example

Points (x,y) \rightarrow sinusoids

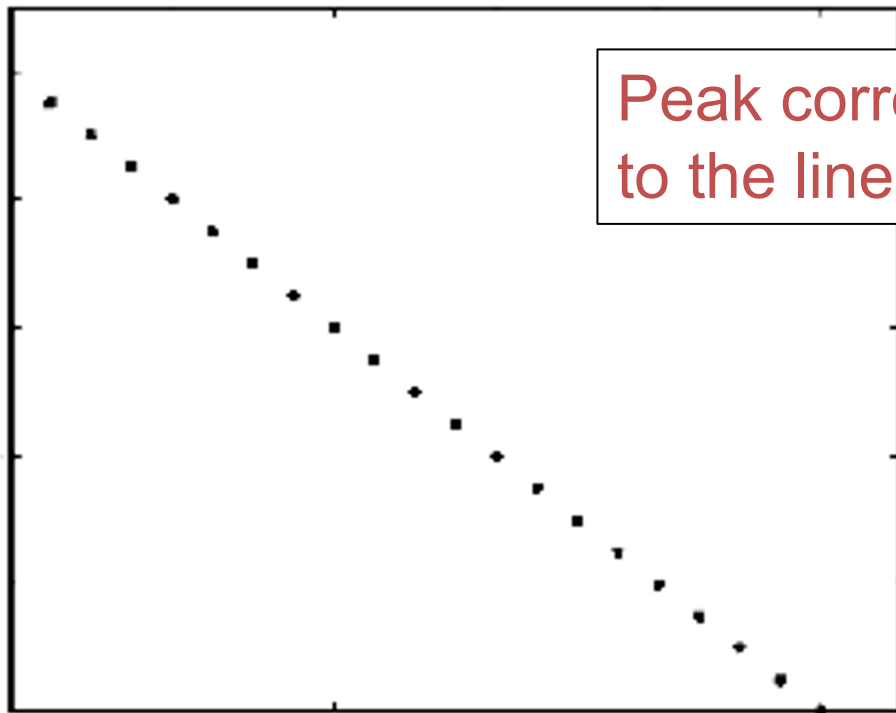
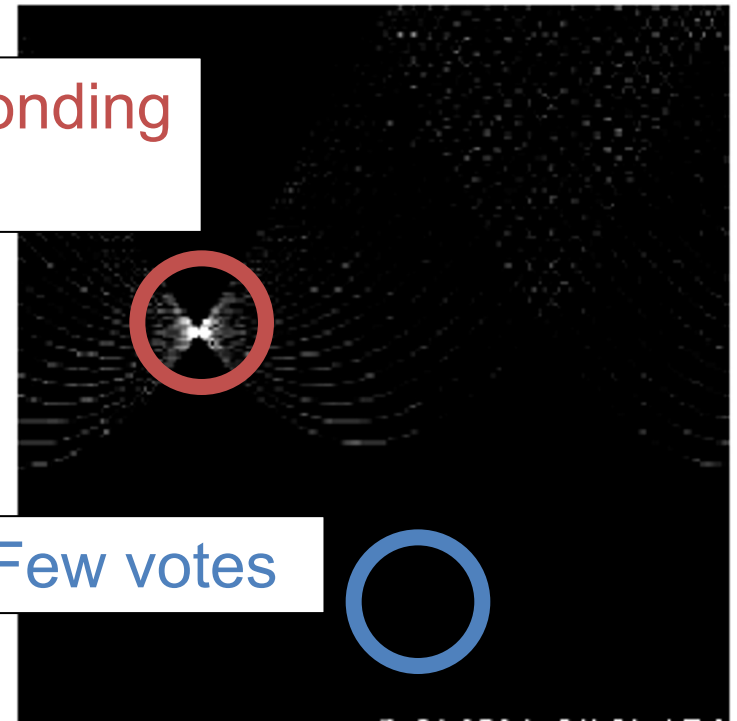


Image Space

Peak corresponding
to the line



Few votes

Parameter Space

Hough Transform Pros / Cons

Pros

1. Handles **multiple** models
2. Some robustness to noise
3. In principle, general

Cons

1. Have to bin ALL parameters: exponential in #params
2. Have to parameterize your space nicely
3. Details really, really important (a working version requires a lot more than what I showed you)

Next Time

- What happens with fitting more complex transformations?

Details for the Curious

Least Squares

Derivation for the Curious

$$\begin{aligned}\|Y - Xw\|_2^2 &= (Y - Xw)^T (Y - Xw) \\ &= Y^T Y - 2w^T X^T Y + (Xw)^T Xw\end{aligned}$$

$$\frac{\partial}{\partial w} (Xw)^T (Xw) = 2 \left(\frac{\partial}{\partial w} Xw^T \right) Xw = 2X^T Xw$$

$$\begin{aligned}\frac{\partial}{\partial w} \|Y - Xw\|_2^2 &= 0 - 2X^T Y + 2X^T Xw \\ &= 2X^T Xw - 2X^T Y\end{aligned}$$

Total Least Squares

- In the interest of less material better, I'm giving that $d = \mu n$.
- This can be derived by solving for d at the optimum in terms of the other variables.

Solving Total Least-Squares

$$\begin{aligned}\|X\mathbf{n} - \mathbf{1}d\|_2^2 &= (X\mathbf{n} - \mathbf{1}d)^T (X\mathbf{n} - \mathbf{1}d) \\ &= (X\mathbf{n})^T (X\mathbf{n}) - 2d\mathbf{1}^T X\mathbf{n} + d^2 \mathbf{1}^T \mathbf{1}\end{aligned}$$

First solve for d at optimum (set to 0)

$$\frac{\partial}{\partial d} \|X\mathbf{n} - \mathbf{1}d\|_2^2 = 0 - 2\mathbf{1}^T X\mathbf{n} + 2dk$$

$$0 = -2\mathbf{1}^T X\mathbf{n} + 2dk \longrightarrow 0 = -\mathbf{1}^T X\mathbf{n} + dk$$

$$\longrightarrow d = \frac{1}{k} \mathbf{1}^T X\mathbf{n} = \mu n$$

Common Fixes

Replace Least-Squares objective

Let $\mathbf{E} = \mathbf{Y} - \mathbf{XW}$

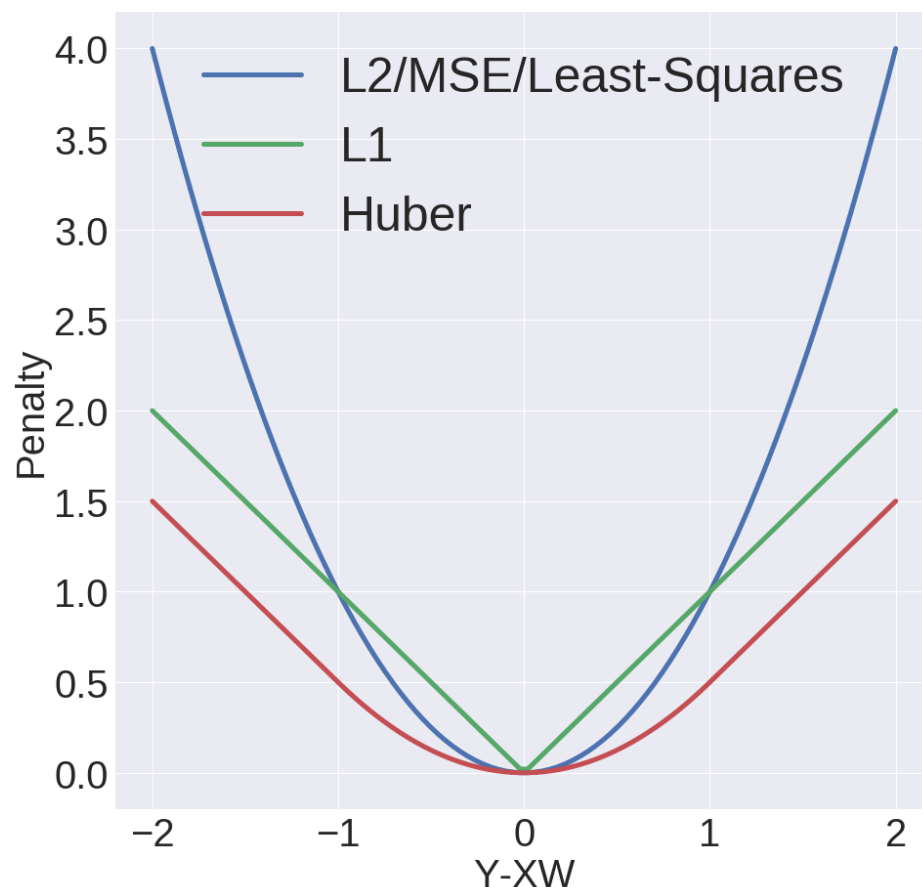
LS/L2/MSE: \mathbf{E}_i^2

L1: $|\mathbf{E}_i|$

Huber:

$$|\mathbf{E}_i| \leq \delta: \quad \frac{1}{2}\mathbf{E}_i^2$$

$$|\mathbf{E}_i| > \delta: \quad \delta\left(|\mathbf{E}_i| - \frac{\delta}{2}\right)$$



Issues with Common Fixes

Why people in vision often don't use other peoples' "robust" methods

- Usually complicated to optimize:
 - Often no closed form solution
 - Typically not something you could write yourself
 - Sometimes not convex (local optimum is not necessarily a global optimum)
- Not simple to extend more complex objectives to things like total-least squares
- Typically don't handle a ton of outliers (e.g., 80% outliers)